

# Alwis documentation

Daniel Fischer  
Benedikt Grundmann

13th March 2006



# Contents

<b>1</b>	<b>GUI manual</b>	<b>7</b>
1.1	The Main Window . . . . .	7
1.1.1	Sessions . . . . .	7
1.1.2	Three Views . . . . .	9
1.1.3	Showing Items . . . . .	10
1.1.4	Finding Items . . . . .	11
1.1.5	Elementary Queries . . . . .	12
1.1.6	Combined Queries . . . . .	13
1.1.7	Query parameters . . . . .	13
1.1.8	Multiple Models . . . . .	14
1.1.9	(Un)weighted Queries . . . . .	14
1.1.10	Most relevant documents for a term query . . . . .	15
1.1.11	Query Engine Processes . . . . .	15
1.2	The Batch Query Wizard . . . . .	16
1.3	The Comparison Window . . . . .	18
1.4	The Measurement Wizard . . . . .	19
1.5	The Config Files . . . . .	19
1.6	Hotkey overview . . . . .	20
1.6.1	Menu hotkeys . . . . .	20
1.6.2	Query hotkeys . . . . .	21

<b>2</b>	<b>Command line tools manual</b>	<b>23</b>
2.1	General . . . . .	23
2.2	Cross platform issues . . . . .	23
2.3	cat2stream . . . . .	24
2.4	filterstream . . . . .	25
2.5	stemstream . . . . .	26
2.6	stream2matrix . . . . .	28
2.7	stream2queries . . . . .	30
2.8	compute-lsi . . . . .	31
2.9	compute-plsi . . . . .	32
2.10	compute-spectral . . . . .	33
<b>3</b>	<b>Format specifications</b>	<b>35</b>
3.1	General . . . . .	35
3.2	Document stream file . . . . .	36
3.3	Stop word list file . . . . .	37
3.4	Base matrix and hold-out matrix files . . . . .	37
3.5	Document list file . . . . .	38
3.6	Term list file . . . . .	39
3.7	Model matrix files . . . . .	39
3.8	GUI config file . . . . .	40
3.8.1	Section PATHS . . . . .	41
3.8.2	Section ENGINE . . . . .	41
3.8.3	Section MDEVICE . . . . .	42
3.9	CDL (Command Description Language) . . . . .	44
3.10	GUI to query engine communication . . . . .	44
3.11	Queries files . . . . .	45
3.12	Qrels files . . . . .	46

<i>CONTENTS</i>	5
3.13 Query results file . . . . .	47
<b>4 Developer's manual</b>	<b>49</b>
4.1 Building on Linux . . . . .	49
4.1.1 Prerequisites . . . . .	49
4.1.2 Building the Alwis suite . . . . .	50
4.2 Building on Windows . . . . .	50
4.2.1 Prerequisites . . . . .	50
4.2.2 Installing Cygwin and MinGW . . . . .	51
4.2.3 Installing Boost C++ libraries . . . . .	52
4.2.4 Installing wxWidgets . . . . .	53
4.2.5 Installing GSL . . . . .	54
4.2.6 Installing ATLAS . . . . .	54
4.2.7 Installing OCaml and tools . . . . .	55
4.2.8 Installing UPX . . . . .	55
4.2.9 Installing MiKTeX . . . . .	55
4.2.10 Building the whole package . . . . .	55
4.2.11 Building the Alwis GUI . . . . .	56
4.2.12 Building the command line tools (C++) . . . . .	56
4.2.13 Building the command line tools (OCaml) . . . . .	56
4.2.14 Building the help files . . . . .	57
4.2.15 Building the installer . . . . .	57



# Chapter 1

## GUI manual

Concept based retrieval schemes are two phase algorithms. They consist of

- a clustering phase in which a model is calculated and
- a query phase in which a calculated model is used to perform queries.

This graphical user interface (GUI) helps you to perform queries and to compare the results from these queries, i.e. the GUI completely focuses on the query phase of concept based retrieval schemes. It does *not* help you to calculate models. In the following chapters we will present the individual parts of the GUI.

### 1.1 The Main Window

A typical search engine interface consists just of a text box in which the query is entered and a window in which the relevant documents are listed. But in order to allow a deeper insight into cluster based retrieval schemes our generic model removes this layer of abstraction. Therefore we have direct access to three entities: documents, terms and concepts. Furthermore there are six query kinds instead of only one: instead of only being able to map a term query to document results you can map from any of the three entities to any of the remaining two entities. It is quite clear that the typical search engine interface is not powerful enough to meet our generic model.

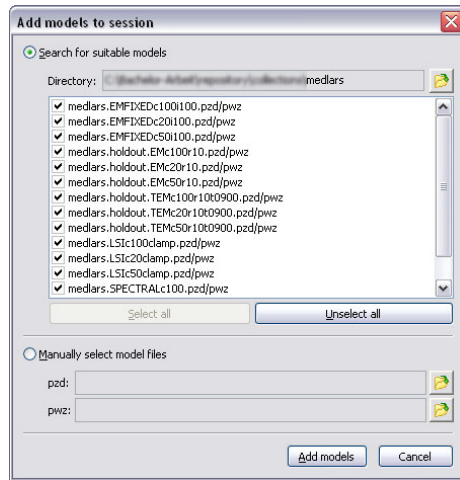
#### 1.1.1 Sessions

In order to start working with Alwis you have to create a session. A session just comprehends all settings needed to work on a specific set of models. One session

always refers to exactly *one* corpus, i.e. one document term matrix (DTM) plus the appropriate document list and term list files, and one or more models. A session can be stored to a file. Loading such a file later on restores all settings for that specific set of models. There can be only one session in one instance of Alwis at a time. But of course you can run multiple instances of Alwis in parallel.

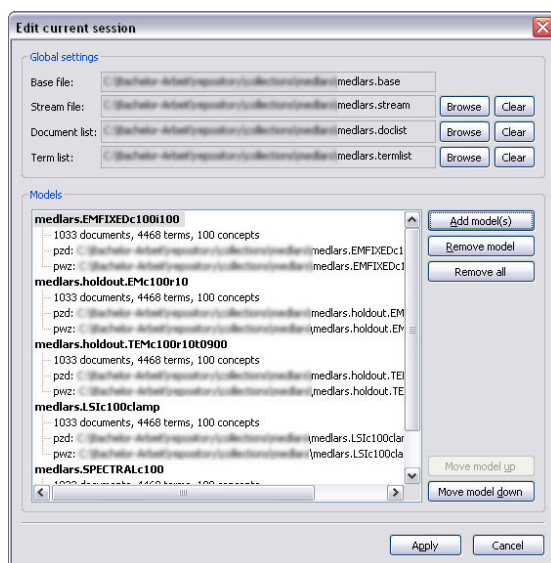
You can create a new session by clicking the “New” button in the tool bar or by clicking the menu item “Session/New session” (keyboard shortcut `Ctrl+N`). Then you will be asked for the appropriate base file, i.e. document term matrix. Alwis tries to infer the paths to the stream file, the document list and the term list automatically. Although these files are optional it is wise to use them in order to be able to show document contents (stream file needed) and to be able to resolve document and term ids (document list and term list needed). If Alwis was not able to infer the paths to these files you can browse for them manually.

Until now your session is empty, i.e. contains no models. You can add models using the button “Add models”. Alwis automatically searches for suitable models and presents them in a new dialog. You just have to select all models you want to visualize and to confirm the dialog. Of course you can search other directories than the default one for suitable models. All models with matching file names are selected by default so that you will just have to confirm the dialog in general.



If the file names of your model files do not match the typical conventions (e.g. `pzd` file base name differs from `pwz` file base name) you can add such a model by manually selecting the `pzd` and `pwz` file. You can add only one model this way at a time. But of course you can call the add models dialog as often as you want.

Remember that all models you are about to visualize *have* to match the selected DTM! This is checked by comparing the number of documents and terms in the DTM with the respective numbers in the models. Obviously this check is not fail-safe. But with carefully chosen file names this should be no problem at all.

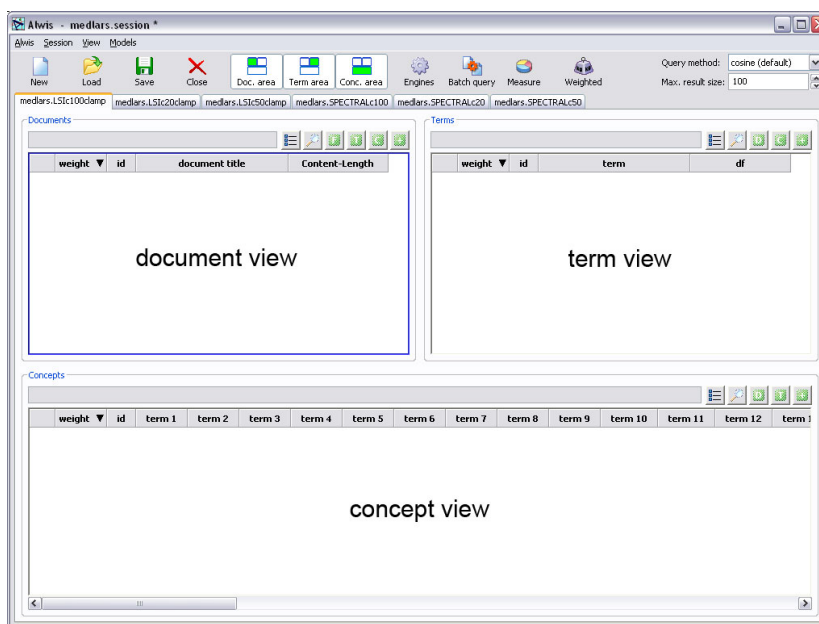


The order of the models in the session dialog equals the order of the models tabs in the main window later on. You can change this order by moving models up or down using the appropriate buttons in the session dialog. The model names (bold face) are used as tab captions as well. You can change them by doing a slow double click on the caption in the session dialog. Then the row turns into a text input field. The changes are committed by hitting return. For obvious reasons there may be no two models with the same name.

Loading and storing sessions should need no explanation at all. Just click on the appropriate tool bar buttons or menu items. You are just asked for the path and file name of the session file. When you are storing a session you will be warned if you try to overwrite an existing file for safety reasons.

### 1.1.2 Three Views

Most space of the main window is taken by three views directly corresponding to the three entities in the generic model: a document view, a term view and a concept view. Each view shows a list of items from the respective entity.



Internally these items are just numbers, namely row or column indices for the matrices a model consists of, with optional weights. Since displaying just these ids would not be very convenient we need to resolve them. The document space as well as the term space are well known (by the document list file and the term list file respectively), therefore we can provide a usable description for document items and term items. Document items are annotated with the document's title and the document's length. On double-clicking a document item the actual contents of that document are shown (if stream file is present). Term items are annotated with the term itself and its document frequency.

Unfortunately cluster based retrieval schemes do *not* provide any semantic description of what a specific concept is. We can only use the implicit information stored in the model to annotate concept items. Therefore concepts are annotated with the most relevant terms for the respective concept.

### 1.1.3 Showing Items

Before you are able to do any queries you will have to show some items in the views initially. Near to each view there is a "item button" (three dots, one below the other, with lines behind them). After clicking such a button a dialog is shown where you can enter an expression which describes what items you want to show in the appropriate view. When typing the expression is continuously checked. The result, i.e. either a error message or the resulting set of items, is shown in the input field just below the input field. The syntax of the expressions is as follows:

- a item can be stated by its id or by its name

- when stating a item by its name you can use clobbering (? matches *one* arbitrary character, \* matches *any* number of arbitrary characters)
- you can state a range of ids by placing a hyphen between the lower id and the higher id
- you can state multiple items by separating them with spaces
- when stating a item name you have to escape all special characters, i.e. ?, \* and space, by prepending a backslash

Here are some example expressions:

- “\*” shows all items
- “1-3” shows the items with the ids 1, 2 and 3
- “1-2 4 6-7” shows the items with the ids 1, 2, 4, 6 and 7
- “foo\*” shows all items with “foo” at the beginning of their name
- “foo bar” shows all items with the names “foo” and “bar”

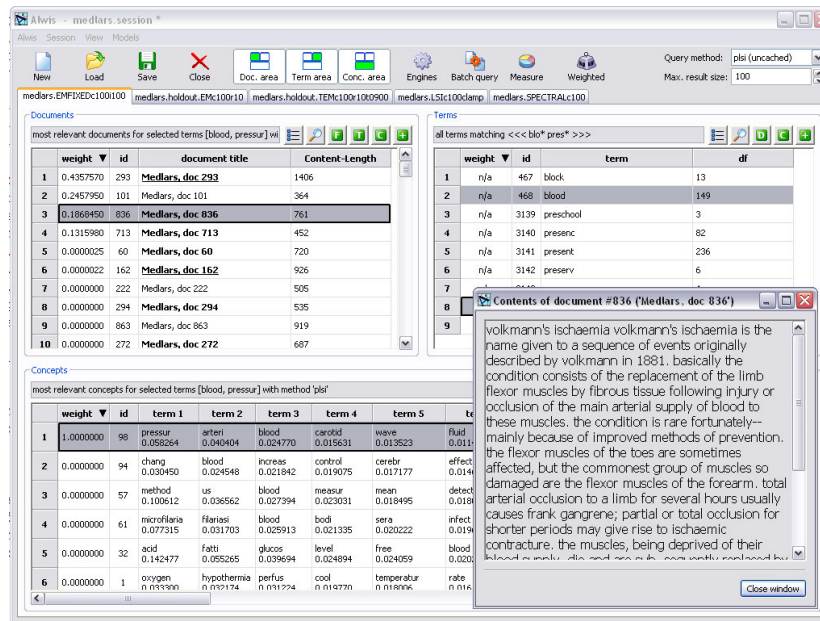
Of course you can mix ids with names in your expression. But remember that concepts do *not* have names! The only useful “name” expression for concepts is “\*” for showing all concepts.

Alternatively you can use hotkeys to trigger the dialog described above. Press `Ctrl+Shift+D` to manually select the documents to show, `Ctrl+Shift+T` for terms and `Ctrl+Shift+C` for concepts.

#### 1.1.4 Finding Items

It can be cumbersome to find a specific item in a query result which can consist of hundreds of items. After clicking the “find button” (which shows a magnifying glass) a dialog is shown where you can enter a search expression. This expression has the same syntax as the one described *above* (section 1.1.3) . All items matching this expression are then highlighted, i.e. selected, in the according view. Furthermore the view is scrolled to the first matching item with respect to the current sort order.

Alternatively you can use the hotkey `Ctrl+F` to open this dialog. You have to set the keyboard focus to the appropriate view first (indicated by a blue border around the view).



### 1.1.5 Elementary Queries

Performing a query is equivalent to mapping items from one entity, i.e. one view, to one of the remaining entities, i.e. one of the two remaining views. In practice this is done by selecting items in one view and clicking the appropriate button near one of the remaining views. In detail there are six elementary queries you can perform:

- most relevant documents for the selected terms**  
 After selecting one or more terms in the term view click the T button near the document view. This is equivalent to pressing **Ctrl+D** while the term view has the keyboard focus.
- most relevant documents for the selected concepts**  
 After selecting one or more concepts in the concept view click the C button near the document view. This is equivalent to pressing **Ctrl+D** while the concept view has the keyboard focus.
- most relevant terms for the selected documents**  
 After selecting one or more documents in the document view click the D button near the term view. This is equivalent to pressing **Ctrl+T** while the document view has the keyboard focus.
- most relevant terms for the selected concepts**  
 After selecting one or more concepts in the concept view click the C button near the term view. This is equivalent to pressing **Ctrl+T** while the concept view has the keyboard focus.

- **most relevant concepts for the selected documents**

After selecting one or more documents in the document view click the D button near the concept view. This is equivalent to pressing **Ctrl+C** while the document view has the keyboard focus.

- **most relevant concepts for the selected terms**

After selecting one or more terms in the term view click the T button near the concept view. This is equivalent to pressing **Ctrl+C** while the term view has the keyboard focus.

### 1.1.6 Combined Queries

Furthermore there are so called combined queries. A combined query actually consists of two elementary queries whose results are combined by multiplying the returned weights of each returned item. That means in detail:

- only items contained in both query results are contained in the final query result
- only items ranked high in *both* query results will be ranked high in the final query result

You can perform a combined query by clicking on the + button near one of the views. If you click e.g. on the + button near the document view the most relevant documents for the selected terms and the most relevant documents for the selected concepts will be queried. The resulting (combined) query result is shown in the document view and can be interpreted as the most relevant documents for the selected terms while limiting the meaning of these terms to that of the selected concepts.

### 1.1.7 Query parameters

Elementary queries as well as combined queries are affected by two parameters: the query method and the maximum query result size. Both parameters can be adjusted using the controls at the right edge of the tool bar. Maybe you have to show the tool bar first by pressing **Ctrl+4** or by clicking the menu item “View/Show tool bar”.

The set of available query methods depends on the query engine you are using. All query methods are specified in the *config file* (section 1.5) . See *here* (section 3.8) for a format description of the config file. By default there are two query methods:

- **cosine** performs simple cosine comparison
- **plsi** performs a generalized folding-in as described in the PLSI algorithm

Query methods can be annotated with either `(default)`, which means that this query method is used for retrieving the most relevant terms for the concepts shown in the concept view, or with `(uncached)`, which means that this is a non-deterministic query method whose results will probably be different each time.

The maximum result size is only used to speed up the graphical user interface. With a maximum result size of e.g. 100 only the 100 most relevant items are shown for each query. Higher values allow you to see less important items as well (or even a complete ranking if the value is high enough). Too high values will result in a poor performance when showing and sorting query results. The default value of 100 is a good compromise for most situations.

### 1.1.8 Multiple Models

This tool has been developed to allow the comparison of different cluster based retrieval schemes. In terms of our generic model this means that we need to compare different models. But what does comparing models actually mean? The most intuitive comparison is to perform the same query on different models and comparing the query results. Therefore we need to concurrently perform queries on different models. Of course switching between the models should require as less effort as possible.

Screen space is technically limited. Therefore it is virtually impossible to show the set of three views each for many models in parallel. Regarding special display devices like beamers it is often even not possible to show two models in parallel. Therefore we have chosen to utilize the tabbed window paradigm. Thereby the working area of the window is enlarged by stacking multiple layers of views. There is one such layer for each model, each containing a document view, a term view and a concept view for the appropriate model. You can easily switch between the layers/models by clicking on the appropriate tab.

### 1.1.9 (Un)weighted Queries

By default all queries are performed unweighted. That means that the selected items will all have the same weight. The weights of the selected items will always sum up to 1. The content of the weight column is *not* considered. Most often this is the wanted behavior.

Sometimes you may want to take the weights from the weight column into account. Each selected item is weighted with the appropriate value from the weight column but the values are normalized so that the weights of all selected items sum up to 1. You can perform weighted queries by activating the weighted query mode by clicking the appropriate button in the tool bar. After the query has been performed the weighted query mode is disabled automatically.

### 1.1.10 Most relevant documents for a term query

Usually one is interested in the most relevant documents for a query consisting of some terms. This kind of query is commonly performed in a special way by cluster based retrieval schemes. First the term query is mapped into concept space and then the most relevant documents for this concept space representation of the query are queried. This is different from the elementary query “most relevant documents for the selected terms” from above! The elementary query described above directly maps from term space to document space instead of mapping to concept space first!

We have made these individual steps explicit in our generic model. But of course this special procedure can be done in the GUI as well by performing the following steps:

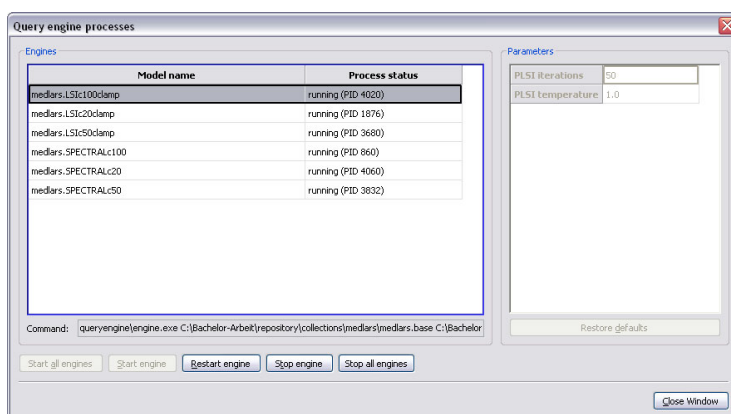
- select the appropriate terms for which you are interested in the most relevant documents
- query the most relevant concepts for the selected terms by clicking on the **T** button near the concept view or by pressing **Ctrl+C** while the term view has the keyboard focus (this step is the actual folding in of the term query into the concept space)
- activate the weighted query modus by clicking the appropriate button in the tool bar or by pressing **Ctrl+W**
- select all concepts in the concept view by setting the keyboard focus to the concept view (e.g. click somewhere in the concept view) and pressing **Ctrl+A**
- query the most relevant documents for the selected concepts by clicking on the **C** button near the document view or by pressing **Ctrl+D** while the concept view has the keyboard focus

For convenience reasons we added a special button **F** which comprehends these five steps into one step. After having selected one or more terms in the term view just click on this button which is located near the document view. The concept view is updated also in order to give you a hint how your term query is mapped into concept space.

### 1.1.11 Query Engine Processes

Queries are executed not by Alwis itself but by a so called query engine. A query engine is an external application which is launched by Alwis. Alwis sends the queries to it and the query engine sends the appropriate results back. Normally you will not have to bother with query engines at all since Alwis handles this transparently for you. It launches a query engine process whenever you open a session or add a model and terminates it again when you close a session or remove a model.

Using the query engine dialog you have some control over the query engine processes. This dialog is showed after clicking the “Engines” button in the tool bar or after clicking the menu item “Models/Query engines” (keyboard hotkey **Ctrl+E**). In this dialog you can (re)start or stop the individual query engine processes. Furthermore you can change their command line arguments (only variable command line arguments). Please note that changing the command line arguments of a query engine is only possible when the query engine is *not* running.



In general you will need this dialog only for solving query engine problems. If a query engine does no longer respond you will probably cancel the query (using the cancel button in the progress dialog shown while the query is being executed). But this cancellation only affects the GUI, it no longer waits for the query result. The query engine itself is still processing this query and therefore it will not respond to future queries as well. You can handle this problem by restarting the query engine in the query engine dialog. Then the external query engine process is forced to terminate and after that it is restarted.

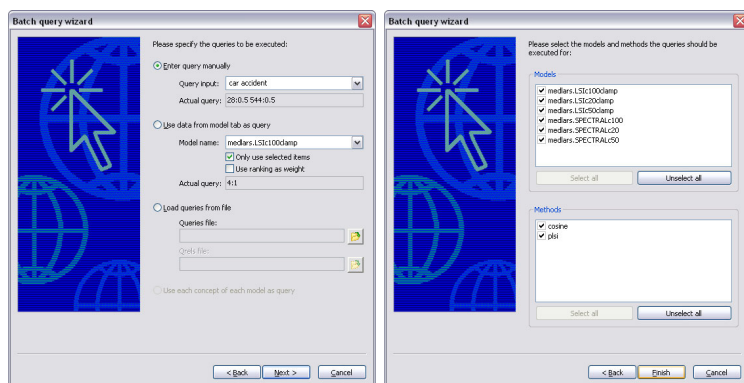
## 1.2 The Batch Query Wizard

Switching between different models in order to find (minor) differences in query results can be quite tedious. Therefore we provide a comparison window which directly sets multiple query results in contrast to each other. Using the batch query wizard you can specify which queries should be executed for which models using which query methods. Furthermore you can specify where the query results should be actually presented:

- Display query results in model tabs:  
The query results are presented in the appropriate views of the appropriate model tabs in the main window. Since there is only one document/term/concept tab for every model this is limited to exactly one query and exactly one query method. But this allows you to do the same query for numerous models easily.

- Display query results in a comparison window:  
The query results are presented in a special comparison window which helps you in comparing the query results from different queries, models and query methods. We will go into detail regarding the comparison window *below* (section 1.3) .
- Save query results to a file:  
The query results are stored to a *query result file* (section 3.13) . This enables you to use an arbitrary external application to present and to compare the query results.

The batch query wizard is shown after clicking on the “Batch query” button in the tool bar or on the corresponding menu item “Models/Batch query wizard” (keyboard shortcut **Ctrl+B**). The dialog itself should be pretty intuitive. So we will not discuss every wizard page in detail here.



You may choose to enter a query by hand (instead of using the selected items of a view and loading queries from a file). The expression you can enter has a similar syntax like the one described *here* (section 1.1.3) . But here you can additionally specify weights for the individual items. Some examples should demonstrate the syntax quite well:

- 1-2:0.25
- 1-2:sum(0.5)
- 1:0.25 2:0.25
- foo:0.8 bar:0.2

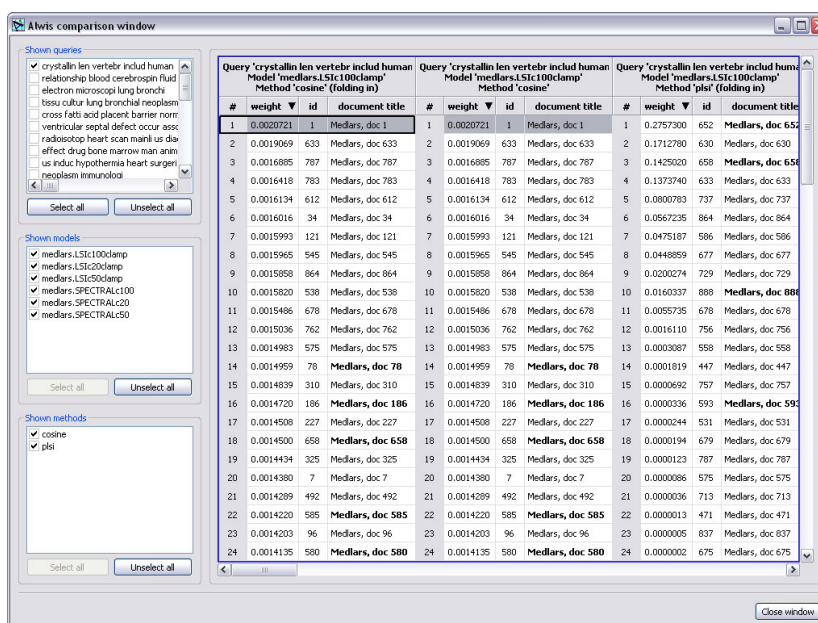
Furthermore there is a special thing about the two last query kinds:

- Concept queries → most relevant documents
- Concept queries → most relevant terms

The concept spaces are introduced by concept based retrieval schemes. Each model introduces its *own* concept space. That means that e.g. concept 1 in one model will not be the same as concept 1 in another model! Therefore we can *not* execute the *same* concept query on *different* models (the query would *not* have the same meaning among different models). The only reasonable thing we can do with concept queries here is to query the most relevant documents/terms for *every* single concept. Thereby we get a kind of “description” for every concept in every model. This could be used to e.g. calculate a 2D “map” of all concepts in all models using self organizing maps (SOM) or a similar algorithm. For obvious reasons these query kinds can *not* be used when displaying the query results in the models tabs.

### 1.3 The Comparison Window

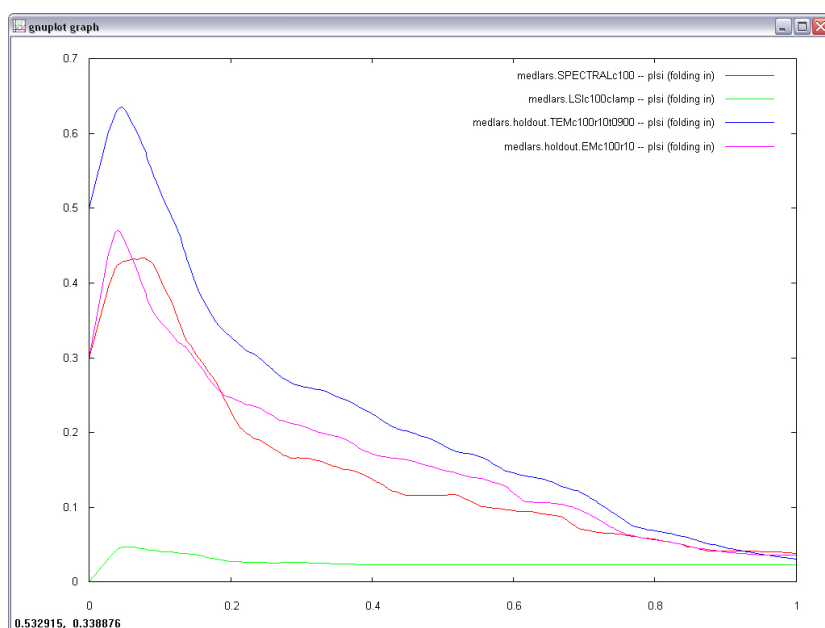
In this window the different query results are stacked in columns which allows an easy comparison. If you select one or more items in one query result the appropriate items in all other query results are highlighted, too. This makes locating corresponding items even more easy. For a better overview you can temporarily hide the query results from specific queries, models or query methods using the check box lists at the left edge of the window. Using the window should be quite intuitive. Therefore we will not go into detail here.



## 1.4 The Measurement Wizard

There are some methods to (semi-)automatically measure models. For instance you can draw a precision recall curve which denotes the precision of query results at different recall levels.

The measurement wizard is very similar to the batch query wizard from *above* (section 1.2) . After specifying the measurement device to use you can specify which queries should be executed for which models using which query methods. After the queries have been executed the results, i.e. in this case the precision recall curve, are presented in a new window.



The selected measurement device determines the kind of the queries which are about to be executed (and if queries need to be executed at all). You are only allowed to enter a query manually or to use the selected items in a view as query if the measurement device does *not* require a *qrels file* (section 3.12) . *qrels* files contain the set of actually relevant documents for every query in the respective *query* file. These sets of actually relevant documents are in general manually determined by humans and represent the “optimal” result a retrieval scheme should achieve.

## 1.5 The Config Files

All parameters needed by Alwis are stored in config files. There is one global config file (applies to all users) and one local config file (applies only to current user). Both config files have the same *format* (section 3.8) and can contain the any settings. The settings made in the local config file have a higher priority

that the settings in the global config file. Probably it is wise to specify *all* required parameters in the *global* config file. The local config file should only be used if you have no rights to write the directory of the global config file or the global config file itself (e.g. you have no administrator privileges).

The global config file is never changed by Alwis. This avoids any problems with missing privileges. Some less important settings, e.g. the last used query method, are stored in the local config file when Alwis terminates. Of course these settings are stored for each user individually.

On Windows the global config file is located at “C:\Windows\Alwis.ini” and the local config file is located at “C:\Documents and Settings\Username\Alwis.ini” (replace `Username` with your user name).

On Linux the local config file is located in the users home directory “`/${HOME}/.Alwis`” and the global config file is located in “`/etc/Alwis.conf`”.

Remember that Alwis can *not* start without a config file or with an erroneous config file. The config files are evaluated when Alwis is loading. Later changes to the config files do *not* have any effects on running instances of Alwis. You will have to restart all running instances of Alwis in order to make the changes take effect.

## 1.6 Hotkey overview

### 1.6.1 Menu hotkeys

**F1** Show help browser.

**Alt+X** Quit application.

**Ctrl+N** Create a new session.

**Ctrl+O** Load a previously saved session.

**Ctrl+S** Save current session.

**Ctrl+1** Toggle document area. Show or hide the area containing the document grid. This has effect on all model pages.

**Ctrl+2** Toggle term area. Show or hide the area containing the term grid. This has effect on all model pages.

**Ctrl+3** Toggle concept area. Show or hide the area containing the concept grid. This has effect on all model pages.

**Ctrl+4** Toggle tool bar. Show or hide the tool bar located on the top of the main window.

**Ctrl+E** Show query engine process dialog. This dialog can be used to achieve a finer control over the external processes used to perform queries.

**Ctrl+B** Show batch query wizard. Execute multiple queries for multiple models with multiple methods. Query results can be visualized in two ways or saved to a file.

**Ctrl+M** Show measurement wizard. Evaluate the quality of the models in the current session using an external program.

**Ctrl+W** Enable weighted query mode. This has only effect to the *next* query being executed since this mode is automatically disabled after query execution.

## 1.6.2 Query hotkeys

**Ctrl+Shift+D** Manually determine the documents to be shown in the document grid on the current model page. You will be prompted for an *expression* (section 1.1.3) specifying the desired documents.

**Ctrl+Shift+T** Manually determine the terms to be shown in the term grid on the current model page. You will be prompted for an *expression* (section 1.1.3) specifying the desired terms.

**Ctrl+Shift+C** Manually determine the concepts to be shown in the concept grid on the current model page. You will be prompted for an *expression* (section 1.1.3) specifying the desired concepts.

**Ctrl+F** Find items in the grid currently having the keyboard focus. You will be prompted for an *expression* (section 1.1.4) specifying the desired items. The specified items will be selected and the grid is scrolled to the first selected item. Any former selection is dismissed.

**Note:** This hotkey only works when either the document grid, term grid or concept grid currently has the keyboard focus. Otherwise this hotkey has no function.

**Ctrl+D** Query most relevant documents for the selected items in the grid currently having the keyboard focus. You may query the most relevant documents for the currently selected terms or concepts depending on which grid has the keyboard focus.

**Note:** This hotkey only works when either the term grid or concept grid currently has the keyboard focus. Furthermore there has to be a non-empty selection in this grid. Otherwise this hotkey has no function.

**Ctrl+T** Query most relevant terms for the selected items in the grid currently having the keyboard focus. You may query the most relevant terms for the currently selected documents or concepts depending on which grid has the keyboard focus.

**Note:** This hotkey only works when either the document grid or concept grid currently has the keyboard focus. Furthermore there has to be a non-empty selection in this grid. Otherwise this hotkey has no function.

**Ctrl+C** Query most relevant concepts for the selected items in the grid currently having the keyboard focus. You may query the most relevant concepts for the currently selected documents or terms depending on which grid has the keyboard focus.

**Note:** This hotkey only works when either the document grid or term grid currently has the keyboard focus. Furthermore there has to be a non-empty selection in this grid. Otherwise this hotkey has no function.

**Ctrl+Alt+D** Perform a combined query in order to get the most relevant documents for the currently selected terms and concepts. You will be prompted for the method to be used to combine the scores of the two atomic queries.

**Note:** There has to be a non-empty selection in the term grid as well as in the concept grid. Otherwise this hotkey has no function.

**Ctrl+Alt+T** Perform a combined query in order to get the most relevant terms for the currently selected documents and concepts. You will be prompted for the method to be used to combine the scores of the two atomic queries.

**Note:** There has to be a non-empty selection in the document grid as well as in the concept grid. Otherwise this hotkey has no function.

**Ctrl+Alt+C** Perform a combined query in order to get the most relevant concepts for the currently selected documents and terms. You will be prompted for the method to be used to combine the scores of the two atomic queries.

**Note:** There has to be a non-empty selection in the document grid as well as in the term grid. Otherwise this hotkey has no function.

**Ctrl+Tab** Switch between the different model pages. Press multiple times to activate the desired model.

**Alt+CursorUp and Alt+CursorDown** Change to the previous/next query method to the selected one. In the upper right corner of the window is a choice box which shows the selected method and allows you to alter it. Using this hotkey you can quickly switch between query methods.

**Alt+Digit** Change to the  $i$ -th query method with  $i$  being the appropriate digit. All keys from **Alt+1** to **Alt+9** are allowed. When trying to change to a non existing query method the last one is selected.

**Alt+CursorLeft and Alt+CursorRight** Increase/decrease the maximum query result length by 10. In the upper right corner of the window is a widget which shows the currently set maximum length and allows you to alter it. Using this hotkey you can quickly change the maximum length.

## Chapter 2

# Command line tools manual

### 2.1 General

These tools do *not* have any graphical user interface or graphical front-end. You have to invoke these programs using a shell. If you are not yet familiar with using shells you should read a tutorial on how to use a shell on your operating system first.

For convenience it is recommended to add the directory containing the binaries to your search path. If there is an installer available on your platform it this should already be done.

### 2.2 Cross platform issues

Unfortunately there are some differences in the way different operating systems handle file names and file contents. For example Linux uses the slash as a file name separator whereas Microsoft Windows uses the backslash. Even more problematic is the fact that Microsoft Windows distinguishes between text and binary files.

These tools have been designed to hide most of these differences. For example the tools accept file names containing slashes instead of backslashes on Windows. Furthermore the files created and read by the tools can be exchanged between different operating systems and platforms.

**Warning:** Pay attention when transferring files created and read by the tools. Some programs like browsers or FTP clients could wrongly classify the files as text files and then change the file contents in a operating system dependent way. Please use *always* binary mode when transferring these files. At best use compressed archives like tar.gz, zip or rar for transferring the files. Compressed

archives ensure the integrity of the enclosed files and reduce the amount of data to be transferred.

## 2.3 cat2stream

With `cat2stream` you can build a document stream from your files. A document stream is used to bundle all files belonging to a corpus. Since such a stream is just one file it simplifies corpus handling and processing.

Type `cat2stream -help` to get a complete list of all options:

```
cat2stream - cat multiple files into a stream file
usage: cat2stream [OPTIONS] INPUT...
```

allowed options:

```
--help           show help message and exit
--version        show program version and exit
-v [ --verbose ] enable verbose mode
-o [ --output ] arg output stream file (default: standard output)
-k [ --keep-empty ] keep empty documents
```

In order to create a document stream just type `cat2stream` followed by the list of files you want to include in the stream. Of course you may use wildcards in the file names.

The resulting document stream is written to standard output per default. If you want to write the stream into a file specify the option `--output` or short `-o` followed by the target file name. If a file with this name already exists it is overwritten.

By default empty documents (file size is zero) are discarded. In order to keep such documents specify the option `--keep-empty` or short `-k`.

Some sample usages:

```
cat2stream *.txt
cat2stream foo*.txt bar*.txt
cat2stream *.txt > foo.stream
cat2stream --output foo.stream *.txt
cat2stream *.txt | filterstream ...
```

Please keep in mind that `cat2stream` does not check for duplicate documents in the stream. If duplicate documents are a potential problem for your purposes you have to take care of that issue yourself.

You may enclose files of any type in your stream. For example you could build a stream from binaries, postscript files and images. Please consider that the other

stream tools expect streams consisting of text files. Since there is no reliable way to check whether a stream only contains text files the results may be unexpected when trying to process a stream containing binary data (although the tools will not crash).

When creating a document stream you should try to convert non plain text files using tools like `ps2text`, `pdf2text`, `html2text`, ... to plain text files before enclosing them in the stream. Files not suitable for text processing like binaries and images should not be enclosed in the stream.

## 2.4 filterstream

`filterstream` combines three processing steps which can be selectively enabled or disabled. With `filterstream` you can

- remove unwanted characters from your stream (these characters are called term delimiter characters since they form the boundaries between the actual terms),
- remove terms shorter than a number of characters you specify and
- remove terms contained in a stop word list file.

Type `cat2stream -help` to get a complete list of all options:

```
filterstream - filter a file stream by chars and terms
usage: filterstream [OPTIONS] [INPUT [OUTPUT]]
```

allowed options:

```
--help          show help message and exit
--version       show program version and exit
-v [ --verbose ] enable verbose mode
-d [ --delim ] arg term delimiters (default: standard delimiters)
-m [ --min-length ] arg minimum term length (default: 2)
-l [ --lowercase ] arg convert terms to lower case? (default: true)
-s [ --stopwords ] arg stop word file (default: none)
-k [ --keep-empty ] keep empty documents
```

With no INPUT or when INPUT is - read from standard input.

With no OUTPUT or when OUTPUT is - write to standard output.

the following characters are always term delimiters:

- all ASCII characters from 0 to 32
- especially space, tab, carriage return and newline

additional standard delimiters are:

- !?@\$%^&^"'+-\_\*=#,;. | () [] {} <> /\0123456789

Text files generally contain a lot of characters interfering with the later processing, e.g. punctuation characters and digits. These characters could form unwanted terms, e.g. numbers, or modify real terms in an unwanted way. Generally all ASCII characters from 0 to 32 are delimiters in stream files. `filterstream` additionally considers some other characters as delimiters. You may specify your own list of additional delimiter characters using the option `--delim` or in short `-d`. In order to disable this feature supply an empty list `--delim ""`.

In most languages nearly all single or two character terms are either meaningless or words which occur so often that they cannot be used to discriminate between different documents. Therefore short terms are often removed. Using the option `--min-length x` or in short `-m x` all terms shorter than `x` characters are removed. By default `x` is 2 and therefore only single character terms are removed.

In most languages capitalization of a term does not contribute to its semantics. Therefore it is often recommended to convert all characters to lowercase. You can enable (value `true`) or disable (value `false`) this feature using the switch `--lowercase` or in short `-l`. The standard C function `tolower` is used for the conversion. Note that certain text file encodings and/or locale settings may lead to unwanted results.

Furthermore there are a lot of terms in every language without usable semantics. Given a file containing a stop word list these terms can be removed from the stream. Depending on the actual application you may want to use a static stop word list or a stop word list containing e.g. the most frequent terms. Using the option `--stopwords` or in short `-s` you can specify the file name of the stop word list to use. For a description of the file format please refer to the Alwis file format reference.

By default empty documents (containing no terms after filtering) are discarded. In order to keep such documents specify the option `--keep-empty` or short `-k`.

Some sample usages:

```
filterstream foo.stream
filterstream foo.stream bar.stream
filterstream --lowercase false foo.stream bar.stream
filterstream --min-length 3 foo.stream bar.stream
filterstream --stopwords english_stop.txt foo.stream bar.stream
cat foo.stream | filterstream > bar.stream
```

## 2.5 stemstream

`stemstream` stems all terms contained in a document stream. This helps to unify terms with the same stem and reduces the noise in the stream.

Type `stemstream -help` to get a complete list of all options:

stemstream - stem all terms contained in the files of a stream  
 usage: stemstream [OPTIONS] [INPUT [OUTPUT]]

allowed options:

```
--help          show help message and exit
--version       show program version and exit
--enum         enumerate supported stemmers/encodings and exit
-v [ --verbose ] enable verbose mode
-l [ --lowercase ] arg convert terms to lower case? (default: true)
-s [ --stemmer ] arg stemmer module to use (default: porter)
-e [ --encoding ] arg character encoding (default: UTF_8)
-k [ --keep-empty ] keep empty documents
```

With no INPUT or when INPUT is - read from standard input.

With no OUTPUT or when OUTPUT is - write to standard output.

There are several stemmer modules available:

```
danish          (allowed encodings: UTF_8, ISO_8859_1)
dutch           (allowed encodings: UTF_8, ISO_8859_1)
english         (allowed encodings: UTF_8, ISO_8859_1)
finnish         (allowed encodings: UTF_8, ISO_8859_1)
french          (allowed encodings: UTF_8, ISO_8859_1)
german          (allowed encodings: UTF_8, ISO_8859_1)
italian         (allowed encodings: UTF_8, ISO_8859_1)
norwegian       (allowed encodings: UTF_8, ISO_8859_1)
porter          (allowed encodings: UTF_8, ISO_8859_1)
portuguese      (allowed encodings: UTF_8, ISO_8859_1)
russian         (allowed encodings: UTF_8, KOI8_R)
spanish         (allowed encodings: UTF_8, ISO_8859_1)
swedish         (allowed encodings: UTF_8, ISO_8859_1)
```

Select the appropriate stemmer module depending on the language of the text files contained in your stream. You may also wish to specify an alternative encoding depending on the encoding of your text files.

stemstream is *not* prepared for multilingual document streams. You may only apply *exactly one* stemmer module to *all* documents in the stream. The same applies to the encoding scheme. If you need multilingual or multi-encoding streams you will have to split them, process the parts separately and join them afterwards.

In most languages capitalization of a term does not contribute to its semantics. Therefore it is often recommended to convert all characters to lowercase. You can enable (value **true**) or disable (value **false**) this feature using the switch **--lowercase** or in short **-l**. The standard C function `tolower` is used for the conversion. The selected encoding does *not* affect the conversion. Note that certain text file encodings and/or locale settings may lead to unwanted results.

By default empty documents are discarded. In order to keep such documents

specify the option `--keep-empty` or short `-k`.

Some sample usages:

```
stemstream foo.stream
stemstream foo.stream bar.stream
stemstream --lowercase false foo.stream bar.stream
stemstream --stemmer spanish foo.stream bar.stream
cat foo.stream | stemstream > bar.stream
```

## 2.6 stream2matrix

`stream2matrix` converts a document stream to matrices needed for further processing, i.e. the calculation of models.

Type `stream2matrix -help` to get a complete list of all options:

```
stream2matrix - convert stream file to sparse matrix file
usage: stream2matrix [OPTIONS] PREFIX [INPUT]

general options:
  --help                show help message and exit
  --version             show program version and exit
  -v [ --verbose ]     enable verbose mode
  -V [ --extra-verbose ] enable extra verbose mode
  -l [ --lowercase ] arg convert terms to lower case? (default: true)
  -m [ --min-docs-per-term ] arg minimum documents per term (default: 2)
  -k [ --keep-empty ]  keep empty documents
  -f [ --force ]       force overwriting of target files

hold-out data options:
  -n [ --hod-nth ] arg use every nth block
  -p [ --hod-prob ] arg use a block with a certain probability
  -b [ --hod-block ] arg number of consecutive terms in a block (default: 10)
```

By default no hold-out data is generated.

With no `INPUT` or when `INPUT` is `-` read from standard input.

The generated files are prefixed with `PREFIX`.

The following files will be created:

`PREFIX.base`, `PREFIX.doclist`, `PREFIX.termlist` and optionally `PREFIX.holdout`

If any of this files exists this is an error unless `--force` is specified.

`stream2matrix` generates up to four files from the document stream:

- a sparse matrix file called “*PREFIX.base*” containing for every combina-

tion of document and term the number of times the term occurs in that document,

- optionally a sparse matrix called “*PREFIX*.holdout” containing the same again but only for the held out part of the stream,
- a document list file called “*PREFIX*.doclist” containing meta data for each document and
- a term list file called “*PREFIX*.termlist” containing meta data for each term.

When invoking `stream2matrix` it is required that none of the target files already exists unless the option `--force` or short `-f` is specified. This also applies to the file “*PREFIX*.holdout” even if no hold-out data is about to be generated.

Often it is useful the omit terms occurring in too few documents. Use the option `--min-docs-per-term` or short `-m` for this purpose. Per default all terms are omitted occurring only in one document.

Hold-out data is used by some algorithms to estimate the quality of a model while calculating it. A certain amount of terms is inserted into the hold-out matrix *instead* of the base matrix. So you will get different base matrices depending on whether hold-out is generated or not. Hold-out data may be generated deterministically or nondeterministically. You may use every *n*-th term block as hold-out data or every term block with a certain probability. For this purpose the document stream is considered as chain of term blocks each containing *m* terms. The dimensions of the base matrix always equal the dimensions of the hold-out matrix.

In most languages capitalization of a term does not contribute to its semantics. Therefore it is often recommended to convert all characters to lowercase. You can enable (value `true`) or disable (value `false`) this feature using the switch `--lowercase` or in short `-l`. The standard C function `tolower` is used for the conversion. Note that certain text file encodings and/or locale settings may lead to unwanted results.

By default empty documents are discarded. In order to keep such documents specify the option `--keep-empty` or short `-k`. Use this option with care since empty documents *may* lead to problems when calculating models depending on the used algorithm.

Even if the option `--keep-empty` is *not* specified the hold-out matrix may (and probably will) contain empty documents. When this option is *not* specified it is only guaranteed that the base matrix does *not* contain empty documents.

Some sample usages:

```
stream2matrix output foo.stream
stream2matrix --min-docs-per-term 5 output foo.stream
stream2matrix --hod-nth 10 --hod-block 5 output foo.stream
```

```
stream2matrix --hod-prob 0.1 output foo.stream
cat foo.stream | stream2matrix output
```

## 2.7 stream2queries

`stream2queries` makes the creation of a queries file more easy. The queries in a queries file have to consist of term ids. When creating such files by hand you would have to lookup the term ids in term list file by yourself. With `stream2queries` you only need to do the following steps:

- create text file containing the queries, one file per query, the file name is used as query description later
- form a stream from this files using `cat2stream`
- apply the same filters to that stream as you did when creating the base matrix
- call `stream2queries` with this process stream and the term list belonging the the base file

Type `stream2queries -help` to get a complete list of all options:

```
stream2queries - convert stream file to queries file
usage: stream2queries [OPTIONS] INPUT TERMLIST [OUTPUT]
```

allowed options:

```
--help          show help message and exit
--version       show program version and exit
-v [ --verbose ] enable verbose mode
-V [ --extra-verbose ] enable extra verbose mode
-l [ --lowercase ] arg convert terms to lower case? (default: true)
-w [ --warn-empty ] enable empty query warnings
```

With no OUTPUT or when OUTPUT is - write to standard output.  
 INPUT is the stream to convert. When INPUT is - read from standard input.  
 TERMLIST is used to translate terms to term ids.  
 Terms no contained in TERMLIST will be discarded.

Each term in the queries stream is searched within the term list. Terms not found will not be used in the queries file. Therefore it may be wise to specify `--warn-empty` or short `-w` or alternatively the extra verbose mode. The warnings may be helpful to discover problems caused by different stream processing.

In most languages capitalization of a term does not contribute to its semantics. Therefore it is often recommended to convert all characters to lowercase. You can enable (value `true`) or disable (value `false`) this feature using the switch

`--lowercase` or in short `-l`. The standard C function `tolower` is used for the conversion. Note that certain text file encodings and/or locale settings may lead to unwanted results.

Some sample usages:

```
stream2queries q.stream foo.termlist
stream2queries q.stream foo.termlist q.queries
stream2queries q.stream foo.termlist > q.queries
```

## 2.8 compute-lsi

`compute-lsi` computes a LSI model from a base matrix file. It just performs a singular value decomposition (SVD) on the base file and postprocesses negative values in the resulting matrices.

Type `compute-lsi -help` to get a complete list of all options:

```
compute-lsi - compute pwz/pzd matrices using latent semantic indexing
usage: compute-lsi [OPTIONS] CONCEPTS DOC-TERM-MATRIX
```

```
-h, --help          show this help and exit
-V, --version       output version information and exit
-v, --verbose       increase verbosity level
-n, --neg           set negative value conversion operator:
                   abs    -- m[i,j] = abs(m[i,j])
                   shift  -- m[i,j] = m[i,j] + min(m)
                   clamp  -- m[i,j] = max (m[i,j], 0) (default)
-o, --output        override output files prefix
-f, --force         overwrite existing files
```

CONCEPTS is the number of concepts the model should distinguish.  
DOC-TERM-MATRIX is the (path and) name of the base matrix file.

You may generate multiple models at a time by specifying more than one concept count and/or more than one negative value conversion operator. Separate the values by commas without spaces in between. Since the SVD has only to be computed once for any number of models this saves a lot of time. A possible command line would look like:

```
compute-lsi -n clamp,abs 20,50,100 sample.base
```

`compute-lsi` generates two matrix files for each model:

- *PREFIX.PARAMS.pzd*
- *PREFIX.PARAMS.pwz*

The *PREFIX* is taken from the name of the base matrix file (anything except the extension *.base*). Using the option `--output` you may override this behavior. *PARAMS* is replaced by the parameters encoded in a human readable form. `compute-lsi` never overwrites existing files unless you specify the option `--force`.

There are three methods available for the conversion of negative values:

- “abs” replaces all negative values with their absolute value,
- “shift” adds a constant value to all matrix cells so that the smallest value is 0.0 and
- “clamp” replaces all negative values with 0.0 (this is the default behavior).

The number of concepts and the negative value conversion operator are only taken into account *after* the SVD has been performed. Therefore any number of models with different parameters can be generated from the result of the SVD. Instead of specifying only one concept count specify a (comma separated) list of concept counts and instead of specifying only one negative value conversion operator specify a (comma separated) list of them. Then for every combination of concept count and negative value conversion operator one model is generated. Since the SVD takes the most time when calculating LSI models this saves a lot of time when calculating a bunch of models.

## 2.9 compute-plsi

`compute-plsi` computes a PLSI model from a base matrix file and optionally a hold-out matrix file. The implemented PLSI algorithms are described by Thomas Hofmann in the paper Probabilistic Latent Semantic Indexing (<http://citeseer.ist.psu.edu>)

Type `compute-plsi -help` to get a complete list of all options:

```
compute-plsi - compute pwz/pzd matrices using the PLSI algorithm
usage: compute-plsi [OPTIONS] CONCEPTS DOC-TERM-MATRIX
```

allowed options:

```
--help          show this help and exit
--version       output version information and exit
-v, --verbose   increase verbosity level
-a, --algorithm PLSI algorithm to use:
                 fixed -- EM with fixed iteration count
                 early -- EM with early stopping
                 tem  -- TEM (default)
-i, --iterations number of iterations (default 100, only 'fixed')
-r, --runs       number of independent runs (default 10, only 'early'/'tem')
-e, --eta        cooling down speed (default 0.9, only 'tem')
```

```

-o, --ouput      override output files prefix
-f, --force      overwrite existing files

```

CONCEPTS is the number of concepts the model should distinguish.  
 DOC-TERM-MATRIX is the (path and) name of the base matrix file.  
 The algorithms 'early' and 'tem' require a hold-out matrix file.  
 The name of this file is inferred from the base matrix file's name.

The three algorithms directly correspond to the three algorithms described in the paper:

- “fixed” performs Expectation Maximization (EM) with a fixed number of iterations,
- “early” performs Expectation Maximization (EM) with early stopping and
- “tem” performs Tempered Expectation Maximization (TEM).

Not all options are relevant for all algorithms. The option `--iterations` is only relevant for the algorithm “fixed” and specifies the fixed number of iterations to be performed. The option `--runs` only applies to the algorithms “early” and “tem” and specifies the number of independent runs to be performed. The option `--eta` is only relevant for the algorithm “tem” and specifies the cooling down speed used for the simulated annealing.

`compute-plsi` generates two matrix files:

- `PREFIX.PARAMS.pzd`
- `PREFIX.PARAMS.pwz`

The `PREFIX` is taken from the name of the base matrix file (anything except the extension `.base`). Using the option `--output` you may override this behavior. `PARAMS` is replaced by the parameters encoded in a human readable form. `compute-plsi` never overwrites existing files unless you specify the option `--force`.

## 2.10 compute-spectral

`compute-spectral` computes a model from a base matrix file using spectral clustering.

Type `compute-spectral -help` to get a complete list of all options:

```

compute-spectral - compute pwz/pzd matrices using spectral clustering

```

usage: compute-spectral [OPTIONS] CONCEPTS DOC-TERM-MATRIX

```

-h, --help          show this help and exit
-V, --version       output version information and exit
-v, --verbose       increase verbosity level
-o, --output        override output files prefix
-f, --force         overwrite existing files

```

CONCEPTS is the number of concepts the model should distinguish.  
 DOC-TERM-MATRIX is the (path and) name of the base matrix file.

You may generate multiple models at a time by specifying more than one concept count. Separate the values by commas without spaces in between. Since the affinity matrix and its eigenvectors have only to be computed once for any number of models this saves a lot of time. A possible command line would look like:

```
compute-spectral 20,50,100 sample.base
```

compute-spectral generates two matrix files for each model:

- *PREFIX.PARAMS.pzd*
- *PREFIX.PARAMS.pwz*

The *PREFIX* is taken from the name of the base matrix file (anything except the extension *.base*). Using the option `--output` you may override this behavior. *PARAMS* is replaced by the parameters encoded in a human readable form. `compute-spectral` never overwrites existing files unless you specify the option `--force`.

The number of concepts is only taken into account *after* the affinity matrix and its eigenvectors are computed. Therefore any number of models with different parameters can be generated from the result of the eigenvector calculation. Instead of specifying only one concept count specify a (comma separated) list of concept counts. Since the affinity matrix and eigenvector calculation takes the most time when calculating Spectral Clustering models this saves a lot of time when calculating a bunch of models.

# Chapter 3

## Format specifications

### 3.1 General

The following chapters specify the file formats and protocols used by the Alwis GUI and command line tools. All these formats and protocols are text based. This has some advantages over binary files:

- content is more or less human readable
- processing with scripts is easier
- some cross platform problems are avoided (like endianness)

Unfortunately some operating systems treat text files and binary files differently. For example on Microsoft Windows some characters and character sequences are altered while reading or writing a file. The applications belonging to the Alwis suite are designed to avoid problems possibly arising from this fact.

*But* you have to pay attention when transferring files e.g. via Internet. Some protocols and transfer applications distinguish between text files and binary files. When files are transferred in text mode the file contents may get altered. Therefore please *always* use binary mode when transferring files created or read by applications from the Alwis suite.

At best always use compressed archives like tar.gz, zip or rar for transferring the files. Compressed archives will always be transferred in binary mode. Furthermore archives ensure the integrity of the enclosed files and reduce the amount of data to be transferred.

## 3.2 Document stream file

A document stream file stores the contents of a variable number of files. The contained files may be of any type. Every file is preceded by a header stating at least its name and its size. The stream file has to satisfy the format **stream**:

```

stream      = document*
document    = block content whitespace*
whitespace = [ \t\r\n]

block       = blockline*
blockline   = assoc | comment
comment     = "#" [^\n]* "\n"

assoc       = key ":" [ \t]* value "\n"
key         = letter ( letter | digit | "-" | "_" )*
value       = [^\n]*
letter      = [A-Za-z]
digit       = [0-9]

```

The header (**block**) preceding each document has to contain at least two associations (**assoc**):

- with key “Path-Name” and value stating the documents name (e.g. path and file name)
- with key “Content-Length” and value being a non-negative integer value stating the document’s length (that’s the size of **content** in bytes)

At the very beginning of the file there may be special comments stating the processing steps performed on the stream until now. These lines start with the character sequence “# \$\$\$ ”. There may be no empty lines before, in-between or after these comment lines.

These special comment lines will not be interpreted by any of the tools. When processing streams they will just be copied to the resulting file(s). These (human readable) comments are intended to give the user a hint by which means the files he is looking at were generated.

The beginning of a stream containing such special comments could look like this:

```

# $$$ cat2stream
# $$$ filterstream min-length 5
# $$$ filterstream stopwords stopwords.txt
Path-Name: foo.txt
Content-Length: ...

```

*Warning:* This file format is quite fragile. The length of the individual documents in the stream is specified in bytes. When the length of a document specified in its header does not equal its actual length the whole stream is broken. When transferring stream files between different computers please take notice of the *general annotations* (section 3.1) .

### 3.3 Stop word list file

A stop word list file contains a list of terms that should be removed from a document stream. In general a stop word is a term that has no usable semantics in a text corpus. The stop word list file has to satisfy the format `stopwords`:

```
stopwords = oneline*
oneline   = term whitespace [^\n]* "\n"
           | [^\n]* "\n"
term      = (^whitespace)+
whitespace = ASCII characters 0 - 32
```

Therefore lines starting with a `whitespace` as well as empty lines are ignored. From all other lines only the part preceding the first `whitespace` is taken as stop word (`term`).

Please note the similarity to the term list file format (see *here* (section 3.6) ). You can use (parts of) term list files as a stop word list file without any change.

### 3.4 Base matrix and hold-out matrix files

The base and hold-out matrix files are stored in a sparse matrix format. The actual data is preceded by a header (`block`). The header contains at least the following three obligatory (integer) values:

- “`rows`” (the number of rows of the matrix)
- “`columns`” (the number of columns of the matrix)
- “`used-cells`” (the number of stored non-zero matrix cells)

The whole file has to fulfill the format `sparsefile`:

```
sparsefile = block whitespace* sparsemat whitespace*
whitespace = [ \t\r\n]

block      = blockline*
```

```

blockline = assoc | comment
comment   = "#" [^\n]* "\n"

assoc     = key ":" [ \t]* value "\n"
key       = letter ( letter | digit | "-" | "_" )*
value     = [^\n]*
letter    = [A-Za-z]
digit     = [0-9]

sparsemat = colstart rowindex values
colptr    = int ( whitespace+ int )*
rowind    = int ( whitespace+ int )*
values    = int ( whitespace+ int )*
int       = digit+

```

In `colptr` there have to be exactly “`columns+1`” integer values, in each `rowind` and `values` exactly “`used-cells`” integer values.

Let’s say we are looking for the stored value in column `C` and row `R`. First we get the indices `i=colptr[C]` and `j=colptr[C+1]` by taking the `C`-th value from `colptr` and its successor. Then we search for the value `R` in `rowind` starting at the `i`-th value in `rowind` and stopping at the `(j-1)`-th value. If we find `R` within this range we take the index `k` where we found it and look for the `k`-th value from `values`. This is the value of the matrix in column `C` and row `R`. Otherwise (we didn’t find `R`) the value is undefined and per default zero.

At the very beginning of both files there may be special comments stating the processing steps performed on the data until now. These lines start with the character sequence “`# $$$`”. There may be no empty lines before, in-between or after these comment lines.

These special comment lines will not be interpreted. These (human readable) comments are intended to give the user a hint by which means the files he is looking at were generated.

### 3.5 Document list file

A document list file contains information about every document a base or hold-out matrix refers to. The information is stored in the form of blocks (`block`) containing associations (`assoc`). In general this information is extracted from the document stream file (see *here* (section 3.2) ). The document list file has to satisfy the format `doclist`:

```

doclist   = block*

block     = blockline*
blockline = assoc | comment

```

```

comment    = "#"  [^\n]*  "\n"

assoc      = key  ":"  [ \t]*  value  "\n"
key         = letter ( letter | digit | "-" | "_" )*
value      = [^\n]*
letter     = [A-Za-z]
digit      = [0-9]

```

In every block the key “id” is associated to the numeric id of the document. The first valid id is 1. The documents in the document list file have to be sorted ascendantly by their numeric id.

### 3.6 Term list file

A term list file contains information about every term a base or hold-out matrix refers to. For every term there is exactly one line. There may be no empty lines or comments. The term list file has to satisfy the format `termlist`:

```

termlist = oneline*
oneline  = term "\t" id "\t" df "\n"
          | term "\t" id "\t" df "\t" garbage "\n"
term     = [^ \t\r\n]+
id       = digit+
df       = digit+
garbage  = [^\n]*
digit    = [0-9]

```

`term` is simply the term itself. `id` is the numeric id of this term starting with 1. `df` states the number of different documents this term occurs in (document frequency).

### 3.7 Model matrix files

The model matrix files (pzd and pwz matrix files) are stored in a dense matrix format. The actual data is preceded by a header (`block`). The header contains at least the following three obligatory (integer) values:

- “documents” (the number of documents)
- “terms” (the number of terms)
- “concepts” (the number of concepts)

The actual dimension of the matrix depend on the file type. In pzd matrix files the matrix has “**concepts**” rows and “**documents**” columns. In pwz matrix files the matrix has “**terms**” rows and “**concepts**” columns.

The whole files have to fulfill the format `densefile`:

```

densefile = block whitespace* densemat whitespace*
whitespace = [ \t\r\n]

block      = blockline*
blockline  = assoc | comment
comment    = "#" [^\n]* "\n"

assoc      = key ":" [ \t]* value "\n"
key        = letter ( letter | digit | "-" | "_" )*
value      = [^\n]*
letter     = [A-Za-z]
digit      = [0-9]

densemat   = int ( whitespace+ int )*
```

Since we need rows times columns values `densemat` has to consist of exactly that number of integer values. The data is stored row major, i.e. first all cells of the first row, then all cells of the second row and so on.

At the very beginning of both files there may be special comments stating the processing steps performed on the data until now. These lines start with the character sequence “`# $$$`”. There may be no empty lines before, in-between or after these comment lines.

These special comment lines will not be interpreted. These (human readable) comments are intended to give the user a hint by which means the files he is looking at were generated.

### 3.8 GUI config file

The syntax of the GUI config file is quite simple. It is a simple text file and can be edited with any text editor.

The file contains one or more sections, each indicated by a section heading enclosed in brackets, e.g. `[sec1]`. Each section can contain an arbitrary number of associations. An association is a line in the form `foo=bar` which associates the key `foo` with the value `bar`. Please note that the backslash `\` is used to escape special characters. To use a backslash, e.g. in Windows path names, you have to type a double-backslash `\\`.

The section headings are treated like paths. If `[sec1]` is a section then `[sec1/subsec]` specifies a subsection `subsec` within the section `sec1`.

Empty lines and lines starting with a hash # will be ignored.

### 3.8.1 Section PATHS

The section `PATHS` specifies the (default) paths for some types of files. This is just for convenience in order to minimize unnecessary directory browsing. The following keys are used:

`session` Default path for loading and storing sessions. If empty or not present the current user's home directory is used as default path.

`collection` Default path for loading base files, document lists, term lists, query files and qrels files. If empty or not present the current user's home directory is used as default path.

`executable` Path which is used as current working directory when external executables (query engines and measurement devices) are invoked.

A typical section `PATHS` looks like:

```
[PATHS]
session=C:\\Programme\\Alwis\\sessions
collection=C:\\Programme\\Alwis\\collections
executable=C:\\Programme\\Alwis\\gui
```

### 3.8.2 Section ENGINE

The section `ENGINE` specifies the query engine to use and its capabilities. Only one key is used:

`cmd` Command line to invoke query engine in *CDL* (section 3.9) format. This command line may contain paths relative to the executable directory specified in the section `PATHS`. This value is mandatory.

The following *CDL* (section 3.9) placeholders are defined:

`#{base}` Path to the base file the query engine is launched for.

`#{pzd}` Path to the pzd matrix file the query engine is launched for.

`#{pwz}` Path to the pwz matrix file the query engine is launched for.

`#{doclist}` Path to the used document list file. This will evaluate to the empty string if no document list is used.

`${termlist}` Path to the used term list file. This will evaluate to the empty string if no term list is used.

Furthermore the section `ENGINE` has to contain one or more subsections specifying the query methods supported by the engine. The name of each subsection is used as name of the method. The subsection names have to be disjunct. The following keys are defined in such subsections:

`desc` Optional text describing the method. Is used as tool tip in the GUI.

`cache` Boolean value stating whether the method's query results are deterministic (value `yes`) or not (value `no`). For deterministic values a query result cache is used to improve performance.

There has to be at least one method. The topmost method in the config file is used as default method and is used for retrieving the concept descriptions. It is strongly recommended that this method is cacheable.

A typical section `ENGINE` looks like:

```
[ENGINE]
cmd=engine.exe ${base} ${pzd} ${pwz} ${?PLSI iterations:int=10}

[ENGINE/cosine]
desc=cosine similarity
cache=yes

[ENGINE/plsi]
desc=folding in from PLSI algorithm
cache=no
```

### 3.8.3 Section MDEVICE

The section `MDEVICE` specifies all available measurement devices. It contains no associations but an arbitrary number of subsections. The names of the subsections are used as the names for the measurement devices. These names have to be disjunct. Each subsection uses the following keys:

`type` Specifies the type of the queries the measurement device evaluates. There are seven possible values for this key:

- `no-query`
- `docs-for-terms`
- `docs-for-concepts`
- `terms-for-docs`

- `terms-for-concepts`
- `concepts-for-docs`
- `concepts-for-terms`

The value “no-query” will prevent Alwis from performing queries. The measurement device can only evaluate the base file and the model files. The other values’ semantics should be obvious.

- `cmd` Command line to invoke measurement device in *CDL* (section 3.9) format. This command line may contain paths relative to the executable directory specified in the section `PATHS`. This value is mandatory.
- `desc` Optional text describing the measurement device. Is used as tool tip in the GUI.
- `qrel` Boolean value stating whether the user should be asked for the path of a `qrels` file (value `yes`) or not (value `no`). Defaults to `no`. The value `yes` is not allowed for `type=no-query`, `type=concepts-for-docs` and `type=concepts-for-terms`.

The following *CDL* (section 3.9) placeholders are defined:

`#{base}` Path to the base file in use.

`#{doclist}` Path to the document list file in use. This will evaluate to the empty string when no document list is used.

`#{termlist}` Path to the term list file in use. This will evaluate to the empty string when no term list is used.

`#{pzd}` Paths to the `pzd` matrix files of all considered models separated by spaces. The order of the paths equals the order of the models in the GUI.

`#{pwz}` Paths to the `pwz` matrix files of all considered models separated by spaces. The order of the paths equals the order of the models in the GUI.

`#{pzdpwz}` Paths to the `pzd` and `pwz` matrix files of all considered models separated by spaces. For each model two paths are inserted, first the `pzd` path and second the `pwz` path. The order of the paths equals the order of the models in the GUI.

`#{qresults}` Path to temporary file containing the results from the executed queries. The measurement device should delete this file upon termination. When no queries were executed (`type=no-query`) this value is empty. For information about the format of this file see *here* (section 3.13) .

A typical section `ENGINE` looks like:

```
[MDEVICE/precision recall]
type=docs_for_terms
cmd=precrecall.exe #{qresults}
desc=generates data files for gnuplot
qrel=yes
```

### 3.9 CDL (Command Description Language)

The CDL is a simple but quite powerful replacement language. All placeholders that may be replaced have the form “`${...}`”. There are placeholders which are replaced automatically by Alwis and there are placeholders which are replaced by user specified values.

There are four possible types of user specified variables:

- `${?name:string=Hello World}`
- `${?name:int=0}`
- `${?name:float=0.4}`
- `${?name:choice=One,Two,Three,Four}`

The character “?” marks a user specified variable. “**name**” is the name of the variable which is presented to the user. Obviously this name should give an idea of the variable’s meaning. The type “**string**” allows an arbitrary character sequence, the type “**int**” only integer numbers, the type “**float**” only IEEE floating point numbers and the type “**choice**” only one of the given comma-separated string options. The default value is specified after the equality sign, in the case of a choice variable the first value is default.

Placeholders with the form “`${name}`” (without leading question mark before “**name**”) are automatically replaced variables. Please refer to the individual format specification (where CDL is used) for defined names. Placeholders with undefined names are *not* replaced.

### 3.10 GUI to query engine communication

The communication between the GUI and the external query engine process uses a simple text protocol. The query engine reads commands from standard input, writes results to standard output and writes status messages to standard error. For every line read from standard input the query engine writes *exactly* one line to standard output (with only one exception). Status messages can be written to standard error at any time and will not be parsed.

The most basic command the query engine has to support is **bye**. When the query engine reads the line **bye** it should terminate gracefully. The command **bye** has no result and therefore the query engine should not write anything to standard output. It is recommended that the query engine terminates with exit code 0.

There are six other commands the query engine has to support. All these commands have to comply with the following syntax:

```

command = kind "[" method "]" " count query
kind    = "docs-for-terms"
        | "docs-for-concepts"
        | "terms-for-docs"
        | "terms-for-concepts"
        | "concepts-for-docs"
        | "concepts-for-terms"
method  = [^]*
count   = [0-9]+
query   = ( weightid " " )+
weightid = id ":" weight
id       = [0-9]+
weight  = IEEE floating point number

```

`kind` specifies the type of query to be performed. The semantics of the possible values stated above should be obvious. `method` is a string specifying the query method to use. The actual value is selected by the user among all possibilities enumerated in the *config file* (section 3.8). `count` specifies the number of most relevant results the user wants to retrieve. `query` contains the actual query in the form a weighted id vector.

### 3.11 Queries files

A queries file stores an arbitrary number of queries. The file has to fulfill the format `queries`:

```

queries = query*
        | header query*
header  = block
query   = block

block   = blockline*
blockline = assoc | comment
comment = "#" [^\n]* "\n"

assoc  = key ":" [ \t]* value "\n"
key    = letter ( letter | digit | "-" | "_" )*
value  = [^\n]*
letter = [A-Za-z]
digit  = [0-9]

```

Each query block has to contain at least two associations. The value for the key `desc` states a description for the query and the value for the key `query` states the actual query. The description is any character sequence. The query is a space-separated list of integer numbers (list of id numbers).

The optional header specifies the type of the contained queries (term queries, document queries or concept queries). The `header` block is recognized by the existence of the key “`type`”. If the value of this key is “`document`” the file contains document queries, the value “`term`” means term queries and the value “`concept`” means concept queries. When there is no header the queries contained in the file are assumed to be term queries.

The `header` block should *not* contain the keys “`desc`” and “`query`” as well as the `query` blocks should *not* contain the key “`type`”.

At the very beginning of both files there may be special comments stating the processing steps performed on the data until now. These lines start with the character sequence “`# $$$` ”. There may be no empty lines before, in-between or after these comment lines.

These special comment lines will not be interpreted. These (human readable) comments are intended to give the user a hint by which means the files he is looking at were generated.

**Note:** Concept queries files are not used/supported by Alwis.

## 3.12 Qrels files

A `qrels` file stores relevant results for queries stored in a queries file. The file has to fulfill the format `qrels`:

```

qrels    = qrel*
          | header qrel*

header   = "document\n"
          | "term\n"
          | "concept\n"

qrel     = int  spacetab  int  "\n"
          | int  spacetab  int  spacetab  garbage  "\n"

digit    = [0-9]
int      = digit+
spacetab = [ \t]
garbage  = [^\n]*

```

The optional `header` determines what kind of relevant results are stored in this file (relevant documents, relevant terms, relevant concepts). When the header is missing the file is assumed to contain the relevant documents.

Each `qrel` line contains two id numbers. The first is the index number of the according query in the queries file. The second is the item id of *one* relevant

item (e.g. relevant document) for this query. There may be any number of lines with the same query index number but different item id numbers (in general there is more than one relevant item for a query).

**Note:** Concept qrels files are not used/supported by Alwis.

### 3.13 Query results file

A query result file stores the results of one or more queries which were performed for one or more models with one or more methods. The file has to fulfill the format qresfile:

```

qresfile = block qdescs queries models methods qresults

qdescs   = qdesc* "\n"
qdesc    = int " " qdesc1 "\n"
qdesc1   = [^\n]*

queries  = query* "\n"
query    = int " " query1 "\n"
query1   = int ":" weight
          | int ":" weight " " query1

models   = model* "\n"
model    = int " " model1 "\n"
model1   = [^\n]*

methods  = method* "\n"
method   = int " " method1 "\n"
method1  = [^\n]*

qresults = qresult* "\n"
qresult  = int " " int " " int " " qresult1 "\n"
qresult1 = int ":" weight
          | int ":" weight " " qresult1

block    = blockline*
blockline = assoc | comment
comment  = "#" [^\n]* "\n"

assoc    = key ":" [ \t]* value "\n"
key      = letter ( letter | digit | "-" | "_" )*
value    = [^\n]*
letter   = [A-Za-z]
digit    = [0-9]
int      = digit+
weight   = IEEE floating point number

```

The header `block` contains the following keys:

- the key “`query-type`” specifying the type of the executed queries (“`documents`”, “`terms`” or “`concepts`”)
- the key “`result-type`” specifying the type of the query results (“`documents`”, “`terms`” or “`concepts`”)
- the key “`base-file`” specifying the absolute path to the base file
- the key “`doc-list`” specifying the absolute path to the document list file
- the key “`term-list`” specifying the absolute path to the term list file
- the key “`pzd-files`” specifying a list of absolute paths to all pzd matrix files uses while querying (separated by spaces, paths containing spaces are enclosed in quotation marks)
- the key “`pwz-files`” specifying a list of absolute paths to all pwz matrix files uses while querying (separated by spaces, paths containing spaces are enclosed in quotation marks)
- the key “`qrel-file`” specifying the absolute path to the qrel file (optional)

The section `qdescs` contains descriptions for every query that has been executed. These description are useful for labeling tables and graphs. The section `queries` contains the executed queries in machine readable format. Both sections must have the same number of lines. The integer values at the beginning of each line start at 1 and indicate the index number of the query.

The section `models` contains the names of the models the queries were executed for. The section `methods` contains the names of the methods the queries were executed with. Both are useful for labeling tables and graphs. The integer values at the beginning of each line start at 1 and indicate the index number of the model or of the method respectively.

The section `qresults` contains the actual query results. At the beginning of each line there three index numbers. The first one is the query index number, the second one the model index number and the last one the method index number. Therefore it is possible to associate the actual query results with the query parameters. Please note that it is in general *not* necessary that there is a `qresult` line for every possible combination of query, model and method index numbers.

# Chapter 4

## Developer's manual

### 4.1 Building on Linux

#### 4.1.1 Prerequisites

- A suitable recent version of GCC (tested with 4.0.2)
- Boost C++ libraries. This includes boost-regex, boost-program-options and the boost-test library which some distributions pack into separate packages.
- wxWidgets  $\geq 2.6.0$  (it does *not* work with older versions)  
Note that it was only tested with the `gtk` version of wxWidgets. Unfortunately Alwis only works with the non unicode version of wxWidgets. At the time Alwis was developed we were unable to get a unicode version running on both Windows and Linux.
- GNU Scientific Library (GSL)
- Automatically Tuned Linear Algebra Software (ATLAS)
- OCaml including its build tools (aka findlib)
- latex (only for manual in PDF format)
- UPX (Ultimate Executable Packer)

As the installation procedure for the packages mentioned above differs widely across the different Linux distributions, we will not explain it here. Please refer to your distributions manual or ask your system administrator.

Please note that many distributions do not include the `tex2rtf` tool in their wxWidgets packages. If you want to use make use of the help facility you must acquire the sources to `tex2rtf` from the wxWidgets website and compile it yourself. As mentioned above you probably have to do so anyway, as many distributions only pack the unicode version of wxWidgets.

### 4.1.2 Building the Alwis suite

There is a global makefile in the directory “src”. There are five targets in this makefile: “gui”, “tools\_gcc”, “tools\_ocaml”, “help” and “setup”. The target “setup” only works on windows (it builds the installer). In order to build the whole package on Linux just type the following commands:

```
make gui
make tools_cpp
make tools_ocaml
make help
```

At the end of the compilation process the executables are still located at their corresponding source directories as listed below. You should copy them into a directory in your path.

Executable	Location
cat2stream	src/tools/stream/cat2stream
filterstream	src/tools/stream/filterstream
stemstream	src/tools/stream/stemstream
stream2queries	src/tools/stream/stream2queries
stream2matrix	src/tools/stream/stream2matrix
compute-spectral	src/tools/compute-spectral/compute-spectral
compute-lsi	src/tools/compute-lsi/compute-lsi
compute-plsi	src/tools/compute-plsi/compute-plsi
engine	src/queryengine/engine
alwis	src/gui/alwis

All you need to do now is adapt the Alwis configuration file and copy it into the standard places. We provide a template configuration file at “src/linux/Alwis.conf”, which should work out of the box. Please refer to the *corresponding section* (section 3.8) for further information.

## 4.2 Building on Windows

### 4.2.1 Prerequisites

- Windows 2000/XP or higher (it *may* work with older versions)
- Cygwin environment
- recent version of MinGW
- Boost C++ libraries
- wxWidgets >= 2.6.0 (it does *not* work with older versions)
- GNU Scientific Library (GSL)

- Automatically Tuned Linear Algebra Software (ATLAS)
- OCaml with some tools
- UPX (Ultimate Executable Packer)
- MiKTeX (only for manual in PDF format)
- Inno Setup Compiler (only for installer)

### 4.2.2 Installing Cygwin and MinGW

Go to the Cygwin website (<http://www.cygwin.com/>) and download the setup tool. This tool helps you to download the actual packages. The default package selection is almost fine. You will only need the following additional packages:

- Archive / zip
- Devel / binutils
- Devel / gcc
- Devel / gcc-core
- Devel / gcc-g++
- Devel / gcc-mingw-core
- Devel / gcc-mingw-g++
- Devel / make
- Devel / mingw-runtime
- Devel / ocaml

For detailed information on how to install Cygwin please refer to the Cygwin website.

Then go to the MinGW website (<http://www.mingw.org/>) and download the packages to install MinGW. You will need the following packages:

- gcc-core
- gcc-g++
- mingw-runtime
- mingw-utils
- w32api
- binutils

You will not need MSYS since you have installed Cygwin. It is recommended to download the binary packages (`.tar.gz` or `.exe` instead of `-src.tar.gz`). Then installing MinGW is just unpacking to a new directory on you hard disk (while maintaining the directory structure contained in the archives). For more detailed information on how to install MinGW please refer to the MinGW website.

### 4.2.3 Installing Boost C++ libraries

Download Boost from the Boost website (<http://www.boost.org/>). You will need the actual Boost package (source) and the Boost Jam package (binary NTx86). Unpack the first package to any location on you harddisk and unpack the file `"bjam.exe"` from the second package to a directory contained in your search path (e.g. `"C:\MinGW\bin"`).

Open a shell and change into the directory you have unpacked the Boost package to. Type the following command:

```
bjam "-sTOOLS=mingw" --prefix=C:\Boost install
```

Change into the directory `"C:\Boost\include\boost-version"`. There is a directory `"boost"` containing the Boost headers. Move this directory into the include directory of MinGW with following command:

```
mv boost C:\MinGW\include
```

Now change into the directory `"C:\Boost\lib"`. Since gcc does not support `#pragma` comments we have to rename the compiled libraries when copying them to the MinGW lib directory. Just execute the following commands:

```
mv libboost_date_time-mgw-s.lib C:\MinGW\lib\boost_date_time.lib
mv libboost_filesystem-mgw-s.lib C:\MinGW\lib\boost_filesystem.lib
mv libboost_iostreams-mgw-s.lib C:\MinGW\lib\boost_iostreams.lib
mv libboost_prg_exec_monitor-mgw-s.lib C:\MinGW\lib\boost_prg_exec_monitor.lib
mv libboost_program_options-mgw-s.lib C:\MinGW\lib\boost_program_options.lib
mv libboost_regex-mgw-s.lib C:\MinGW\lib\boost_regex.lib
mv libboost_serialization-mgw-s.lib C:\MinGW\lib\boost_serialization.lib
mv libboost_signals-mgw-s.lib C:\MinGW\lib\boost_signals.lib
mv libboost_test_exec_monitor-mgw-s.lib C:\MinGW\lib\boost_test_exec_monitor.lib
mv libboost_unit_test_framework-mgw-s.lib C:\MinGW\lib\boost_unit_test_framework.lib
mv libboost_wave-mgw-s.lib C:\MinGW\lib\boost_wave.lib
```

After that you can delete the directory you unpacked the boost sources to.

### 4.2.4 Installing wxWidgets

Assume you have installed Cygwin to “C:\Cygwin” and MinGW to “C:\MinGW”. Ensure that the directories “C:\MinGW\bin” and “C:\Cygwin\bin” are contained in you search path, at best at the very beginning of you search path. “C:\MinGW\bin” has to precede “C:\Cygwin\bin”.

You may change your search path on Windows 2000/XP with the following steps:

- right click on “My Computer” and select “Properties”
- go to the tab “Advanced”
- click the button “Environment variables”
- double click the variable “PATH” (category “System variables”)
- prepend “C:\MinGW\bin;C:\Cygwin\bin;” to the existing value

After changing environment variables you have to close shells (command line prompts) and reopen them in order to make the changes take effect.

Download wxWidgets from the wxWidgets website (<http://www.wxwidgets.org/>). Alwis has been developed with the version 2.6.2. It is incompatible with older versions (less than 2.6). It should work with newer versions but it may require changes to the source code. Unpack the downloaded file to “C:\”, a new directory called “wxWidgets-2.6.2” will be created.

Open a shell and change into your wxWidgets directory. If you plan to compile a debug version of Alwis you will need the debug version of wxWidgets. Execute the following commands to build it:

```
mkdir build-debug
cd build-debug
bash ../configure --with-msw --enable-html --enable-debug
                    --enable-debug_gdb --disable-shared
                    --prefix=C:/wxWidgets

make
make install
```

For building the release version of Alwis you will only need the release version of wxWidgets. Execute the following commands to build it:

```
mkdir build-release
cd build-release
bash ../configure --with-msw --enable-html --disable-debug
                    --disable-shared --prefix=C:/wxWidgets

make
make install
```

You may execute both sequences of commands if you need both versions of wxWidgets. There are no file name clashes between the two versions.

Now change to directory “`utils/tex2rtf`” within your wxWidgets directory and execute the following commands:

```
make
cd src
strip -s -v tex2rtf.exe
upx --best -v tex2rtf.exe
```

Copy the resulting file “`tex2rtf.exe`” to a directory contained in your search path, e.g. “`C:\MinGW\bin`”. It is needed to compile the help files.

### 4.2.5 Installing GSL

Download the GNU Scientific Library installer from the GSL website (<http://gnuwin32.sourceforge.net>) and install it into the same directory you have installed MinGW into, e.g. `C:\MinGW`. It copies some header files to the “`include`” directory and some pre-compiled library files to the “`lib`” directory. That’s all.

### 4.2.6 Installing ATLAS

You can download the ATLAS precompiled library from two different locations:

- the ATLAS sourceforge page (<http://math-atlas.sourceforge.net/>) offers the precompiled library for Linux (works with MinGW as well)
- the old netlib page (<http://www.netlib.org/atlas/archives/windows/>) offers the precompiled library for Windows

Any version greater or equal 3.0 should suffice. There are several packages optimized for different CPUs. You will have to trade off between performance and compatibility. The SSE2 package will probably be the fastest but will not work with older processors. The Alwis executables are build using the SSE1 optimized version and therefore work with all Pentium class processors.

In the downloaded archive (regardless of version and optimization level) contains some header files (`*.h`) and some precompiled libraries (`*.a`). Sometimes the archive contains a directory structure. The paths stored in the archive should *not* be used. Unpack the header files to your MinGW’s `include` directory (e.g. “`C:\MinGW\include`”) and the library files to your MinGW’s `lib` directory (e.g. “`C:\MinGW\lib`”). Remember *not* to use the paths stored in the archive. That’s all.

### 4.2.7 Installing OCaml and tools

The easiest way to install OCaml on Windows is to download the Ocaml-MinGW-Maxi Distribution from this website (<http://lasagne.unix-ag.uni-kl.de/omm/>). Just unpack the downloaded archive to “C:\”, a new directory “ocamlmgw” is created. This package contains everything you need, especially the OCaml compiler itself and the tool `ocamlfind`.

You need to adapt your search path when compiling OCaml programs (at least temporarily). For more information see the *OCaml build notes* (section 4.2.13).

### 4.2.8 Installing UPX

Installing the Ultimate Executable Packer is as simple as possible. Just download the precompiled Windows version from the UPX website (<http://upx.sourceforge.net/>) and unpack the contained executable “`upx.exe`” to any directory contained in your search patch.

### 4.2.9 Installing MiKTeX

Download MiKTeX setup from the MiKTeX website (<http://www.miktex.org/>). There are three different packages offered by the setup. The smallest one should suffice. After installing ensure that the MiKTeX bin directory is included in your search path. You can check this by typing “`pdflatex -version`” in a shell.

### 4.2.10 Building the whole package

There is a global makefile in the directory “`src`”. There are five targets in this makefile: “`gui`”, “`tools_gcc`”, “`tools_ocaml`”, “`help`” and “`setup`”. In order to build the whole package just type the following commands:

```
make gui
make tools_cpp
make tools_ocaml
make help
make setup
```

Probably you will need to check your search path settings before invoking “`make tools_ocaml`”. For further details see the *OCaml build instructions* (section 4.2.13). For more details on how to build the single components of Alwis please refer to the following sections.

### 4.2.11 Building the Alwis GUI

The Alwis GUI consists of two components: the core (directory “`src/core`”) and the actual GUI (directory “`src/gui`”). You have to build the core first. Change into the core directory and type the following commands:

```
make clean
make FINAL=1
```

Now change into the gui directory and type the following commands:

```
make clean
make FINAL=1
```

Now there should be a file called “`alwis.exe`”. You may omit the parameter “`FINAL=1`” when building the core and gui in order to get the debug version of Alwis. **Note:** The debug version of Alwis is quite huge - about 70 MB.

When having problems building the GUI you should check the settings in the GUI's makefile. Maybe you have to adapt some settings like the path to your wxWidgets directory.

### 4.2.12 Building the command line tools (C++)

The command line tools written in C++ are located in the directory “`src/tools`”. Just change into this directory and type the following commands:

```
cd stemmer
make clean
make
cd ..
make clean
make FINAL=1
```

Now there should be five executables called “`cat2stream.exe`”, “`filterstream.exe`”, “`stemstream.exe`”, “`stream2matrix.exe`” and “`stream2queries.exe`”. Again you may omit the parameter “`FINAL=1`” in order to get the debug versions of these tools.

### 4.2.13 Building the command line tools (OCaml)

Assume you have installed Cygwin to “`C:\Cygwin`” and OCaml to “`C:\ocamlmgw`”. Ensure that the directories “`C:\ocamlmgw\bin`” and “`C:\Cygwin\bin`” are contained in you search path, at best at the very beginning of you search path.

**Important:** Now the directory “C:\Cygwin\bin” has to precede the directory “C:\MinGW\bin” in your search path. There is an incompatibility between the OCaml compiler and the non-Cygwin version of MinGW. If you get strange errors while compiling you should check your search path for this order.

Change into the directory “src/queryengine”. Just type following commands to build the query engine:

```
make
make pack
```

Now change into the directory “src/tools/compute-plsi”. Again just type the following commands to build the tool for calculating PLSI models:

```
make
make pack
```

And finally change into the directory “src/mdevices”. Once again just type the following commands to build the precision recall measurement device:

```
make
make pack
```

#### 4.2.14 Building the help files

For building the help files just change into the directory “src/help” and type “make”. Then there is a new folder “alwis\_help” containing plain HTML files (e.g. for the use as online manual), a new file “alwis\_help.htb” used by the built-in Alwis help browser and a new file “manual.pdf” for viewing and printing with a PDF viewer.

#### 4.2.15 Building the installer

You will need the Inno Setup Compiler from the Inno Setup website (<http://www.innosetup.com>). Furthermore all components of Alwis should have been built (preferably in FINAL mode, stripped and compressed). Simply open the file “setup.iss” from the directory “src/setup” with the Inno Setup Compiler. Select “Compile” from the menu “Build”. It will create a file called “alwis-version.exe” in the directory “src”.