

# Emdros Query Tool User's Guide – Version 3.3.0

Ulrik Petersen

July 4, 2011

## Contents

### 1 Introduction

This is a short User's Guide to the Emdros Query Tool (aka eqt).

The Emdros Query Tool reads MQL queries and gives back the results in a way that makes sense in the context of *your* database.

#### 1.1 Origins of the query tool

The original algorithms for the Emdros Query Tool were written by **Hendrik Jan Bosman** in Python. Thus he is the real father of the Emdros Query Tool. The algorithms were rewritten in C++ by Ulrik Petersen.

#### 1.2 This Guide

This User's Guide is divided into four parts:

1. This introduction, which explains how to get started,
2. Graphical version User's Guide
3. Configuration of the program
4. Query Guide (including the MQL Cheat Sheet)

#### 1.3 PCRE Library

Regular expression support is provided by the PCRE library package, which is open source software, written by Philip Hazel, and copyright by the University of Cambridge, England.

PCRE can be downloaded from:

<ftp://ftp.csx.cam.ac.uk/pub/software/programming/pcre/>

## 1.4 Three versions

The Emdros Query Tool exists in three versions:

- A command-line version (**eqtc**)
- Two graphical versions:
  - A non-Unicode version (**eqt**)
  - A Unicode-aware version (**eqtu**)

The next page explains how to get started with either of these versions.

## 1.5 Getting started

### 1.5.1 Introduction

Before running the Emdros Query Tool for the first time on a database, you need to write a configuration file that matches your database. This is a one-off thing: Once it's done, you don't need to bother with it any more.

### 1.5.2 Sample configuration file

A number of sample configuration files are supplied with Emdros. You can use these as a starting point for writing your own configuration file.

One supplied configuration file is called "default.cfg", while another is called "wihebrew.cfg". You can search for these on your computer to locate where they are, or see the manual page for eqt to know where they are installed (on Windows, they are installed in

### 1.5.3 Full details

The "default.cfg" file is almost self-documenting. However, you can get more information about the details of the configuration file here:

- Configuring the program

## 1.6 Command-line version

After you've written the configuration file, you can then proceed to running the Emdros query tool, like this:

```
eqtc -c <your-config-file> myquery.mql
```

For example:

```
eqtc -c myconfigfile.cfg myquery.mql
```

### 1.6.1 Saving the output

The results will be printed on standard output, so you can redirect that to a file for later viewing:

```
eqtc -c <your-config-file> myquery.mql > myoutput.txt
```

### 1.6.2 More options

If using MySQL or PostgreSQL, you may need to pass a password to the program. Do this with the `-p` option. You may also need to pass a database username (`-u`) or the name of the database host computer (`-h`). For example:

```
eqtc -c <your-config-file> -u <dbuser> -p <password> -h <dbhost> myquery.mql
```

To get a list of supported options, run `eqtc` with the `--help` switch:

```
eqtc --help
```


## 2 Graphical version User's Guide

This section of the Emdros Query Tool User's Guide shows how to use the graphical versions of the Emdros Query Tool.

### 2.1 Starting the program

#### 2.1.1 Getting started

Once you open the program, you will be presented with the main screen. You will then need to "connect" to a database. Either choose the menu-

item "Tools|New database connection" or press the button  "Connect to database".

#### 2.1.2 Connection dialog

You will then be given a dialog box allowing you to choose the Connection Settings. At the top is a drop-down box allowing you to choose the backend. Based on this choice, the dialog box will appear slightly differently depending whether the backend is:

- SQLite, or
- MySQL or PostgreSQL

### 2.1.3 Non-Unicode vs. Unicode

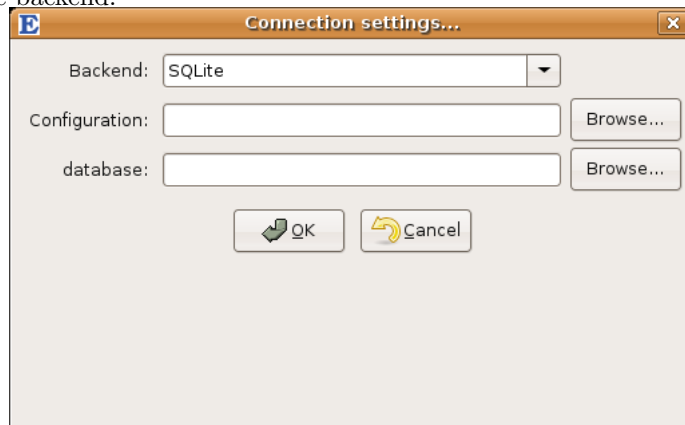
For each backend, the program exists in two versions:

- A non-Unicode version (**eqt**)
- A Unicode-aware version (**eqtu**)

Note that on Linux/Unix, eqtu may in fact be called eqt, replacing the non-Unicode version. This occurs if the installed wxWidgets uses Unicode by default.

### 2.1.4 SQLite version

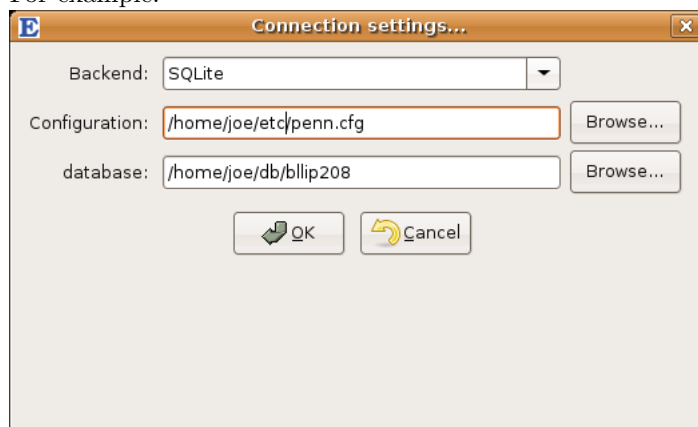
The Connection Settings dialog looks like this if you have selected SQLite as the backend:



**Set the configuration file** The first thing you should do is press the "Browse" button next to the "Configuration" edit box, then navigate to where you have your configuration file.

Once you've opened the configuration file, the "database" field will be filled from the "database" value stored in the configuration file, if any. If this is not the database you want, simply enter (or browse for) the database you want.

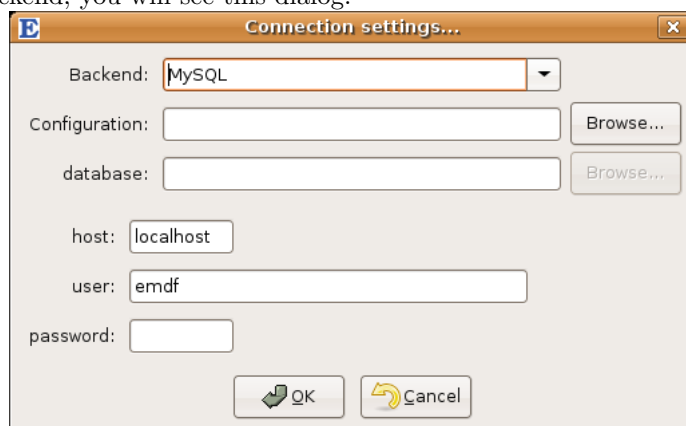
For example:



**Press OK** Once you're done setting the configuration file and the database, press "OK". If you want to quit the program instead, press "Cancel".

### 2.1.5 MySQL/PostgreSQL version

When you start the Emdros Query Tool using the MySQL or the PostgreSQL backend, you will see this dialog:



**Set the configuration file** The first thing you should do is press the "Browse" button next to the "Configuration" edit box, then navigate to where you have your configuration file.

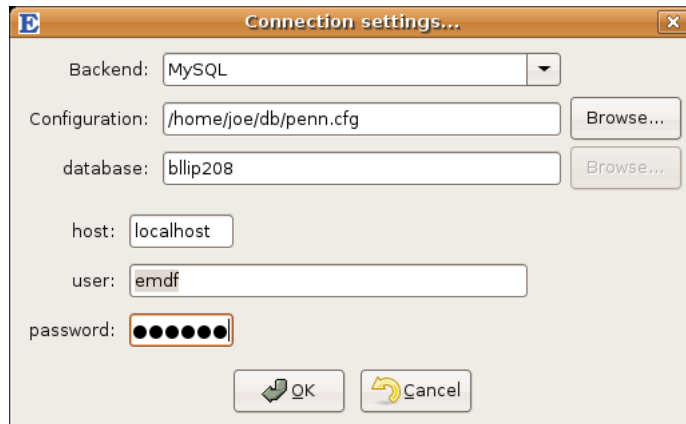
**Database** Once you've opened the configuration file, the "database" field will be filled from the "database" value stored in the configuration file, if any. If this is not the database you want, simply enter the database you want.

**Host, user, password** Most people can leave the "host" and "user" fields as they are, and simply write the password.

The "host" field shows which computer to connect to, i.e., the computer where the MySQL or PostgreSQL backend is running. "localhost" means the computer where eqt(u) is running.

The "user" field is the database user to connect to the backend as. Note that this may be different from your computer user name. The default is "emdf", since that is the recommended default user to create when you bootstrap the MySQL or PostgreSQL database (see "bootstrapping.txt" in the Emdros documentation).

The "password" field is for the password of the database user to connect as. This is set either by the database administrator, or by the one who bootstrapped the MySQL or PostgreSQL database.



**Example**

**Press OK** Once you're done setting the configuration file and the database, press "OK". If you want to quit the program instead, press "Cancel".

## 2.2 The main screen

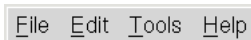
Once you've pressed "OK" on the "Connection settings" dialog box, you will see the main screen:



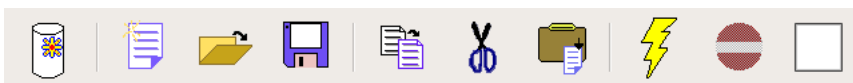
### 2.2.1 Parts

The main screen consist of these parts:

- A menu at the top.



- Below that, a toolbar with buttons.



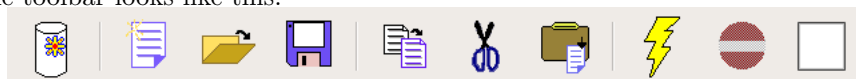
- Below that, three areas:
  - To the right, a collapsible tree that shows the database schema
  - In the top half of the left side, the input area, where you write your queries and your configuration files.
  - In the lower half of the left side, the output area, where the output from queries will be written.

## 2.2.2 Next


Next, we describe each of these parts.


### 2.2.3 Toolbar


The toolbar looks like this:





The buttons represent actions, each of which will be explained below.


**New, Open, Save**  : Clears the input and output areas to "start afresh".


**Open**  : Opens a file (a query or a configuration file).


**Save**  : Saves the current file.

**Copy, Cut, Paste**  : Copies the current selection to the clipboard. Works for both the input area and the output area.

**Cut**  : Cuts the current selection to the clipboard. Works for both the input area and the output area.

**Paste**  : Pastes the current clipboard contents. Works for both the input area and the output area.

**Execute, Stop**  : Executes the query in the input area.

**Stop**  : Stops the current execution. You may have to wait a bit before the execution stops. Please try pressing this button again if execution does not stop immediately.


## 2.2.4 Menus

There are four menus, each explained on its own page:

1. File menu
2. Edit menu
3. Tools menu
4. Help menu

### File menu


**New...** The "New..." menu item clears the input area and the output area, thus "starting afresh".

Equivalent toolbar button:  .

**Open** The "Open" menu item opens a file and reads it into the input area. The file can be either an MQL query or a configuration file.

Equivalent toolbar button:  .

**Save** The "Save" menu item saves the current contents of the input area. If no filename has been given previously, the "Save As" dialog box will appear so you can give the file a name.

Equivalent toolbar button:  .

**Save as** The "Save as" menu item saves the current contents of the input area, giving it a (new) name.


Equivalent toolbar button: None.

**Exit** The "Exit" menu item quits the program.


Equivalent toolbar button: None.

### Edit menu


**Copy** The "Copy" menu item copies the current selection to the clipboard.

Equivalent toolbar button:  .

**Cut** The "Cut" menu item cuts the current selection to the clipboard.

Equivalent toolbar button:  .

**Paste** The "Paste" menu item pastes the current contents of the clipboard.

Equivalent toolbar button:  .


## Tools menu

**New Connection...** The "New Connection..." menu item closes the current database connection and brings up the Connection settings dialog (SQLite version, MySQL/PostgreSQL version).

This allows you to connect to a new database, or to reconnect if the connection was lost, or to use a different configuration file.

Equivalent toolbar button: None.

**Execute query** The "Execute query" menu item attempts to execute the contents of the input area as an MQL query against the backend. The output area will show the results.

Equivalent toolbar button:  .

**Configure...** This menu-item has not been implemented yet. Sorry.

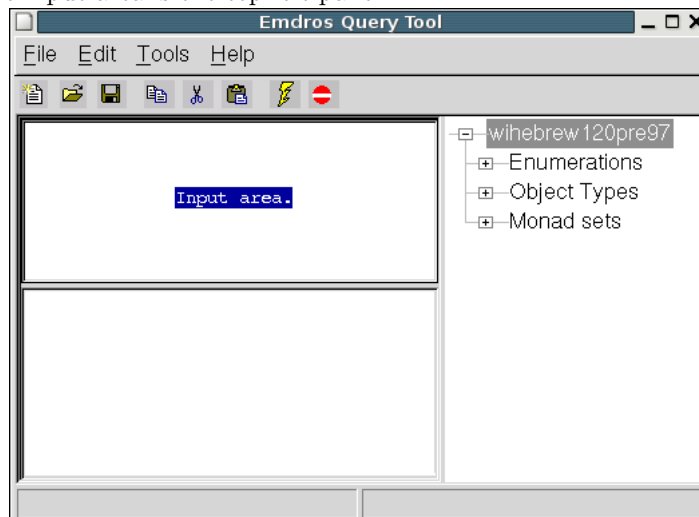
## Help menu

**About Emdros Query Tool...** This brings up the "About box". Press "OK" to dismiss it again.

Equivalent toolbar button: None.

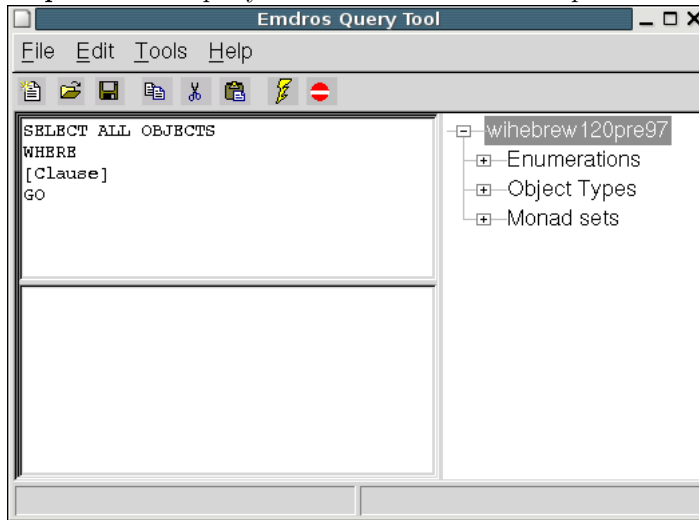
### 2.2.5 Input area

The input area is the top left pane:



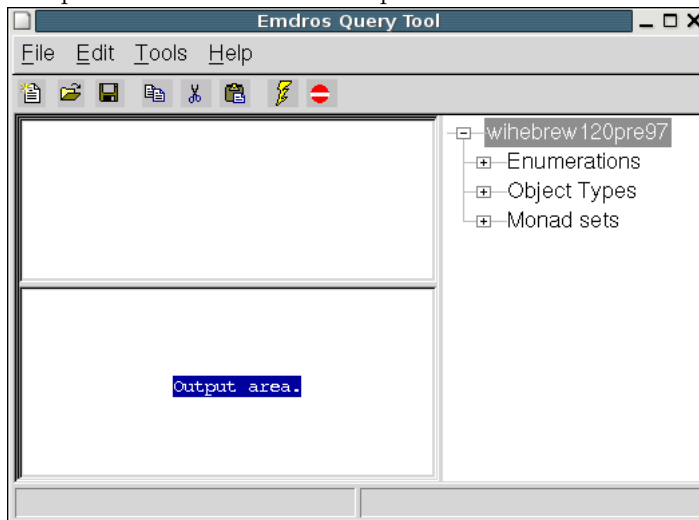
Use it to write/edit your MQL query, or to write/edit a configuration file.

**Example** Here a query has been written in the input area:

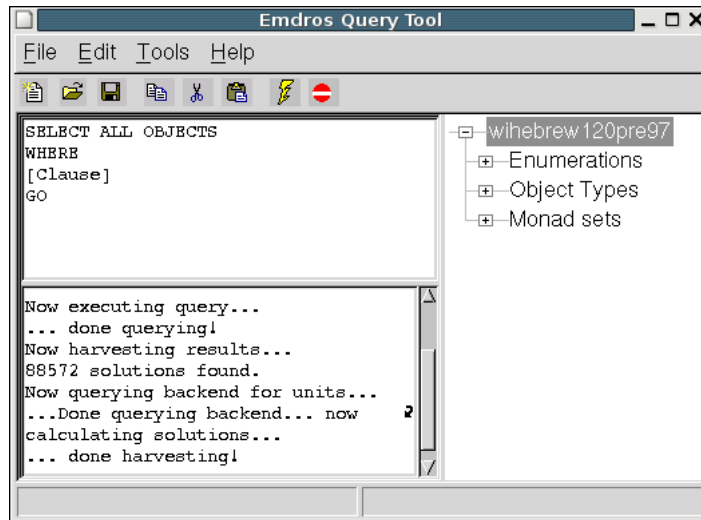


## 2.2.6 Output area

The output area is the bottom left pane:



**Example** Here a query has been executed, and the output area shows the progress of the query. Of course, the output itself would also be shown if one scrolled the window.



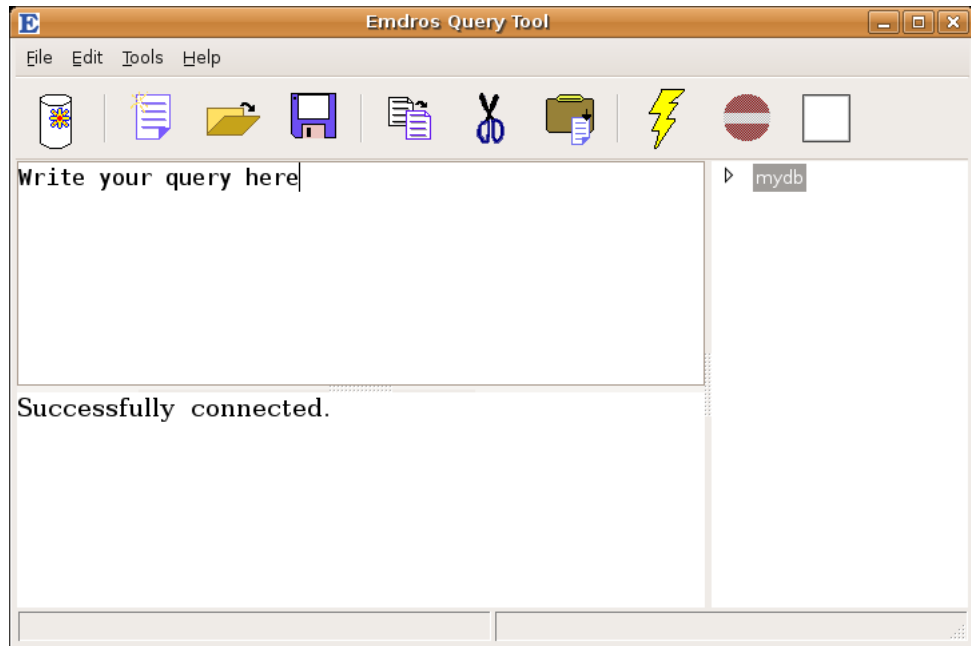
### 2.2.7 Database Schema area

**What is a schema?** The "schema" of an Emdros database consists of the enumerations, object types, and monad sets created as "meta-data" in the database.

An object type could be, e.g., "Word", "Phrase", "Clause", "Verse", "Line", etc., and the designer of the database has determined these when he or she created the Emdros database.

An enumeration is a set of labels that can be used for easy mnemonic access to commonly used values. Examples could include "noun", "verb", "adjective", etc. for the enumeration "part\_of\_speech".

**The schema area** The pane in the right-hand side shows the database schema of the current database in a tree.



**The schema tree** The schema area consists of a tree which is fully expandable/collapsible by pressing the "+" and "-" buttons next to a label.

The "root" of the tree shows the name and/or location of the current database. Underneath the root, there are three categories:

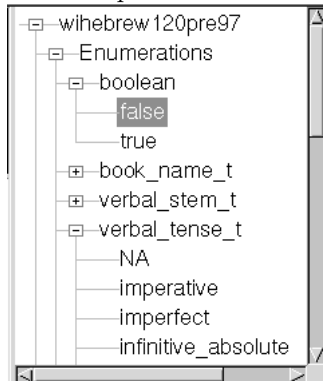
1. **Enumerations**
2. **Object types**
3. **Monad sets**

These will be dealt with in turn.

**Enumerations** An enumeration is a set of labels for easy use of mnemonic values.

The "Enumerations" top-level category can be expanded to show the enumerations available in the current database.

For example:



The *enumerations* appear at the level just below the "Enumerations" label. That means, in this picture, "boolean", "book\_name\_t", and "verbal\_stem\_t" are all enumerations.

You can expand or collapse each enumeration to show or hide its enumeration labels. In the above picture, "true" and "false" are labels in the "boolean" set, whereas "NA", "imperative", "perfect", etc. are labels in the "verbal\_tense\_t" enumeration.

## Object types

**Crash course in object types** An object type groups objects with similar characteristics. For example, the database designer may have created the object types "Word", "Phrase", "Clause", "Book", etc.

An object type has a number of "attributes", also known as "features". The features that an object type has determines the values that can be associated with objects of the given type.

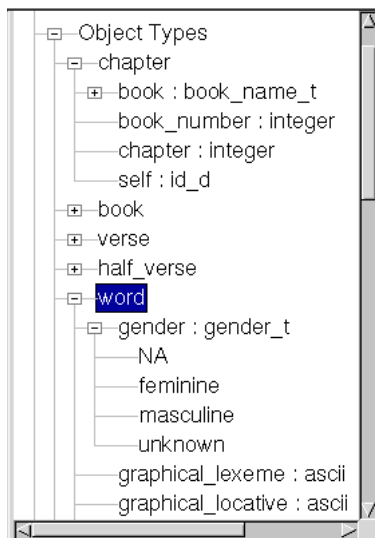
For example, an object of type "Phrase" may have values for features such as "phrase\_type", and "function". This is up to the database designer to decide.

A feature always has a type, drawn from the following table:

- integer: A positive or negative integer, or 0.
- string: An 8-bit string
- ascii: A 7-bit string
- id\_d: A pointer that points to the "self" feature of another object.
- *everything else*: An enumeration.

All objects have a "self" feature which is a unique ID. Other objects can refer to a given object by the value of that object's "self" feature. Database designers use the "id\_d" feature type for this purpose.

**The schema tree** Here is a partial schema tree showing the object types of a sample database:



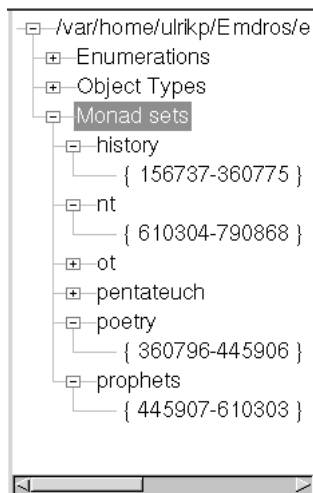
The object types appear directly below the "Object Types" label. For example, in the above picture, "chapter", "book", "verse", and "word", are all object types.

You can collapse or expand each object type by pressing the "+" and "-" buttons next to the object type's name. When expanded, the *features* will appear, along with their type (see the list above).

If a feature's type is an enumeration, then the feature label can be expanded to show the enumeration labels. In the above example, the object type "chapter" has a feature "book" whose type is the enumeration "book\_name\_t". The enumeration labels are not visible, but the user could click the "+" next to "book" to expand the tree and see the enumeration labels.

The object type "word" has a feature "gender" which is of the enumeration type "gender\_t". This has been expanded to reveal the labels "NA", "masculine", "feminine", and "unknown".

**Monad sets** The user can create "arbitrary, named monad sets" in the database by using the "CREATE MONAD SET" query. These can be viewed with the schema tree:



The names of the monad sets appear just below the "Monad Sets" label in the tree.

Each monad set name can be expanded to reveal the monad set in question.

It is just coincidence that the above sets are all single-range sets: The monad sets can be quite arbitrary, and may therefore have gaps.


**Usage** The main benefit of having arbitrary monad sets is that they can be used to limit searches, like this:

```
SELECT ALL OBJECTS
IN Poetry
WHERE
[Clause]
GO
```

Here all clauses in the monad set named "Poetry" will be retrieved.

### 2.3 Normal usage

Normal usage involves:

1. Starting the program.
2. Filling out the Connection settings dialog (SQLite version, MySQL/PostgreSQL version). This connects to a database using a certain configuration.
3. Opening a query, or or writing a query in the input area.
4. Executing the query with the  **execute** button.
5. Examining the output in the output area.
6. Repeating from step 3, or quitting the program.

## 3 Configuring the program

### 3.1

### 3.2 Format of the configuration file

The configuration file follows many other Unix and Windows configuration files in that:

- Comments are prefixed by #, and anything from the # to the end of the line is ignored.
- Blank lines are ignored.
- The rest is a number of "key = value" pairs.
- The keys are pre-defined (see below).
- The values are either "quote-enclosed strings" (e.g., "C:\Emdros\mymap.map") or consist of letters, numbers, underscores, and/or dots, optionally followed by a "quote-enclosed string" (e.g., 'word.surfce', 'word.surface.'C:\Documents and Settings\Administrator\teckitmap.map').

When a value has dots that are not enclosed in "quotes", then the strings on either side of the dots are interpreted as subkeys. For example, the value "word.surface" represents the subkey "word" with the value "surface", and the value "word.surface."/home/myname/Blah.map" represents the subkey "word" with the subsubkey "surface", followed by the value "/home/myname/Blah.map".

Here is a sample configuration file, explained bit by bit:

#### 3.2.1 Database selection

```
# database
database = mydb
```

You can specify a database that is always to be used with this configuration file (unless overridden with the -d switch to eqtc).

If using SQLite, you may wish to specify a path. Do so in quotes:

```
# database
database = "C:\Program Files\Emdros\EmdrosSQLite-1.2.0.pre173\db\mydb"
```

#### 3.2.2 Rasterising unit

```
# rasterising unit
raster_unit = clause
```

The Emdros Query Tool operates with a notion of "rasterising unit". That is the unit to be displayed on one line. For example, if your query returns a bunch of words, then, in the example above, all clauses that contains at least one of the words will be fetched and displayed.

There can only be one rasterising unit.

### 3.2.3 Raster context

```
# raster context
raster_context_before      = 10
raster_context_after      = 10
```

The "raster\_unit" can be replaced with "so many monads of context" (before and after a hit). If a raster\_unit is specified, it will take priority. If a raster unit is not specified, then both of the raster\_context\_before / raster\_context\_after values must be present.

### 3.2.4 Data units

```
# data units
data_unit      = clause
data_unit      = phrase
data_unit      = word
data_feature   = word.surface
data_feature   = word.psp
data_feature   = phrase.phrase_type
data_feature   = phrase.function      # You can have more than one
data_unit_name = clause."Cl"
data_left_boundary = phrase.OPEN_BRACKET # Specifies left boundary marker
data_right_boundary = phrase.CLOSE_BRACKET # Specifies right boundary marker
```

The data units are the units to be displayed in each rasterising line. They can be anything, and need not be words.

You must specify which feature(s) to display for each data unit. The feature-names must be prefixed with the name of the data unit plus a dot, as in the example above.

The capitalisation must be exactly the same as the value for the "data\_unit" key. For example, if you said "data\_unit = phrase", then you must also say "data\_feature = phrase.phrase\_type", not "Phrase.phrase\_type".

There can be more than one data unit. If so, they should be specified in the order from largest to smallest (e.g., clause, phrase, word). This will give the "output" output style (see below) a hint as to how to print things in the right order.

You can optionally specify "boundary markers" that will be printed at the left and right boundaries of a unit respectively. The strings to be printed can be taken from the following table:

This string...	Is replaced by... (without the quotes)
SPACE	" "
COMMA	" , "
COMMA_SPACE	" , "
COLON	" : "
COLON_SPACE	" : "
OPEN_BRACE	" { "
CLOSE_BRACE	" } "
OPEN_BRACKET	" [ "
CLOSE_BRACKET	" ] "
OPEN_PAREN	" ( "
CLOSE_PAREN	" ) "
NEWLINE	newline
NIL	""

The "data\_unit\_name" key gives, for a given object type, a string which will appear above all the other data.features (if any). In the above example, the clause unit is given a "Cl" label.

Finally, in the graphical version of the Emdros Query Tool, it is possible to have an interlinear display. The order of the lines in the interlinear display is the same as the data.feature keys. The number of lines is equal to the number of features for the data unit for which the most data.feature keys are given, plus the number of data\_unit\_name keys for that unit.

### 3.2.5 TECKit mappings

```
#surface
data_feature_teckit_mapping = word.surface."e:\TECKit\mymap.map"
data_feature_teckit_in_encoding = word.surface.bytes
data_feature_teckit_out_encoding = word.surface.unicode

# lemma
data_feature_teckit_mapping = word.lemma."e:\TECKit\mymap.map"
data_feature_teckit_in_encoding = word.lemma.bytes
data_feature_teckit_out_encoding = word.lemma.unicode
```

**TECKit** is a tool made by SIL International. It converts between encodings, in particular to and from Unicode. The Emdros Query Tool incorporates TECKit, and you can apply it to any textual feature of any object type.

TECKit works with a so-called "map file" – a text file which you or someone else writes. More information about writing TECKit mappings can be found on SIL's website:

<http://scripts.sil.org/TECKit/>

The Emdros Query Tool needs three pieces of information in order for TECKit to work on a particular feature:

1. The name of the file which holds the mapping. This is given with the key "data\_feature\_teckit\_mapping".
2. The input encoding (encoding of the feature-string): This is given with the key "data\_feature\_teckit\_in\_encoding". The value can be either "bytes" or "unicode" (without the quotes). "bytes" means that TECKit does not convert to UTF-8. "unicode" means it is converted to UTF-8 for display. You should use whatever is used in the map file for input encoding here.
3. The output encoding (encoding to transform into): This is given with the key "data\_feature\_teckit\_out\_encoding". The same meanings and restrictions apply as for the input encoding.

TECKit can not only convert between encodings, but also remove stuff from a string. This can come in handy when you have characters in your feature-strings which you do not wish to display. Again, see the TECKit site on SIL's website for information on how to write a TECKit mapping.

You should give first the object type, then a dot, then the feature-name, then a dot, then the full path to the map file. You probably need to enclose the path in "double quotes".

You can only have one TECKit per feature.

### 3.2.6 Reference unit

```
# reference units
reference_unit      = verse
reference_feature   = verse.book
reference_feature   = verse.chapter
reference_feature   = verse.verse

reference_sep = SPACE # between book and chapter
reference_sep = COMMA # between chapter and verse
```

If you have a unit in your database which somehow identifies the position in the document, or an ID, you can display these units at the left of each line. The canonical example is the Biblical system of book-chapter-verse, but in many corpora, there will be a unit identifying, e.g., which newspaper article something came from.

In the above example, verse is the reference unit, and three features are fetched, namely book, chapter, verse. The order in which they are specified in the configuration file is the order in which they will be emitted.

If there is more than one reference unit feature, you must specify the separators to separate them. In the above example "SPACE" will be emitted between "book" and "chapter", and "COMMA" will be emitted between the chapter and the verse (again, the order matters). See the table above for some possibilities of using special characters.

There can be only one reference unit.

### 3.2.7 Output style

```
#output_style = kwic
#output_style = tree
#output_style = xml
output_style = output
```

Specifies which implementation to use for emitting solutions. Currently, three kinds of output style are implemented:

- **output**: A "bracketed" view.
- **tree**: A "tree" view
- **kwic**: A "key words in context" view.
- **xml**: XML output view. For usage, please see the DTD that is emitted along with the output.

### 3.2.8 Data tree parent

```
# Tree parent feature.
# If output_style = tree, then it is assumed that
# there is a feature on all relevant data units which gives the
# id_d of the parent. That is, each child node in the tree
# must have a feature which provides the id_d of its parent.
# If a data_unit is provided which does not have a data_tree_parent,
# then that data_unit *must* contain the top-most nodes in the tree.
data_tree_parent = clause.parent
data_tree_parent = phrase.parent
data_tree_parent = word.parent
```

If "output\_style" is set to "tree", then this option specifies, for each terminal and non-terminal in the tree, what feature gives the parent of the node. Note that this feature must have type "id\_d", and the value must point to the id\_d of the parent node.

### 3.2.9 Tree terminal unit

```
# Tree terminal unit.
# If output_style = tree, then the Emdros Query Tool needs to know
# which object types are terminals (i.e., leaf nodes in the tree)
# and which object types are non-terminals. This is done by
# designating *one* (1) data_unit to be the data_tree_terminal_unit.
# The rest of the data_units will then be non-terminals.
data_tree_terminal_unit = word
```

This options tells the tree layout code which data\_unit contains the terminals. Note that the Emdros Query Tool assumes that terminals and nonterminals are different object types. There may be more than one nonterminal object type, but only one terminal object type. The non-terminal object types are determined based on the data\_unit option.

### 3.2.10 Hit type

```
# hit type
# hit_ must be one of:
#   focus
#   innermost
#   innermost_focus
#   outermost
hit_type    = outermost
```

The hit type determines how the sheaf is interpreted. There are four available options:

- **focus:** Means that an object originating in a block with the FOCUS keyword present will result in one "hit".
- **innermost:** Means that only the innermost MatchedObjects will give rise to hits; one hit per string of blocks in which all matched objects have no descendants (i.e., no inner sheaf).
- **innermost\_focus:** Like innermost, but only those matched objects whose "focus" boolean is set will have their monads included.
- **outermost:** Means that only the outermost MatchedObjects will give rise to hits; one hit per outermost MatchedObject.

If none of these are specified, then "outermost" is assumed as the default.

### 3.2.11 Options

```
# display options
option = apply_focus
option = break_after_raster
option = quiet
option = single_raster_units
```

You can have these options:

Option	Meaning
apply_focus	If set, then those data units which had the "focus" keyword in the original query will be surrounded by {braces} in the output.
break_after_raster	If set, then a newline is emitted after each raster-line. If not set, then the raster-lines are run together.
quiet	If set, then only results will be printed; nothing else. If not set, then things like progress and number of solutions will be printed. If an error occurs, then that will be printed regardless of the status of this option.
single_raster_units	If set, then each raster unit will only ever be printed once. This affects the number of solutions printed: If two solutions each contain the same raster unit, then only one of the solutions will be printed.

### 3.2.12 Display options

```
input_area_font_name = "Arial MS Unicode"
input_area_font_size = 11 # in points
output_area_font_name_1 = "SPIonic"
output_area_font_name_2 = "Courier New"
output_area_font_name_3 = "Times New Roman"
output_area_magnification = 100 # in percent (%)
```

You can set the default font name and font size (in points) for the input area.

You cannot set the font size in points for the output area. Instead, you can set it to a percentage of 12 point. For example, setting `output_area_magnification` to 150 will select a font size of 18 points, and setting it to 200 will select a font size of 24 points.

## 4 Query Guide

### 4.1 Topographic queries vs. table-queries

The Emdros Query Tool can display two kinds of results: Sheaves (from topographic queries) and tables (from all other queries).

### 4.2 Topographic queries

Topographic queries are described in the cheat sheet.

### 4.3 MQL Cheat Sheet

#### 4.3.1 Preamble

For topographic queries, you must prefix the query with this magic incantation:

```
SELECT ALL OBJECTS
WHERE
// Your query here.
```

Variations over this exist:

```
SELECT ALL OBJECTS
IN MyMonadSet
WHERE
// Query here, will only find objects
// in the stored monad set "MyMonadSet"
```

```
SELECT ALL OBJECTS
IN {1-23400}
WHERE
// Query here, will only find objects
// within the monads {1-23400}.
```

#### 4.3.2 Overview

- The basics
- Feature-restrictions
- First/last
- Object references
- NOTEXIST and Kleene Star

### 4.3.3 The basics

#### Object blocks, Sequence, Embedding

Construction	Meaning	Example
[ObjectType]	Objects: Finds object of type ObjectType	[Word]
[A] [B]	Adjacency: Finds objects of type A that are adjacent to objects of type B. However, if there is a gap in the context, that gap will be ignored and objects on either side of the gap will be "adjacent"	[Word] [Phrase]
[A]! [B]	Strict adjacency: Finds objects of type A that are <u>really</u> adjacent to objects of type B. No gaps allowed.	[Word]![Word]
[A [B ]	Embedding: Finds objects of type A inside which there is an embedded object of type B.	[Phrase [Word] ]

#### Arbitrary space

Construction	Meaning	Example
[A [B] .. [C] ]	Arbitrary space: Finds objects of type A, inside of which there are two objects, one of type B and one of type C, and they need not be adjacent (though they can be).	[Clause [Phrase] .. [Phrase] ]
[A [B] .. <= 5 [C] ]	Arbitrary space with restriction: Finds objects of type A, inside of which there are two objects, one of type B and one of type C, and they need not be adjacent (though they can be), and there may be up to 5 monads between them.	[Clause [Phrase] .. <= 20 [Phrase] ]
[A [B] .. < 6 [C] ]	Arbitrary space with restriction: Finds objects of type A, inside of which there are two objects, one of type B and one of type C, and they need not be adjacent (though they can be), and there may be up to 5 monads between them.	[Clause [Phrase] .. < 21 [Phrase] ]
[A [B] .. BETWEEN 3 AND 6 [C] ]	Arbitrary space with restriction: Finds objects of type A, inside of which there are two objects, one of type B and one of type C, and they need not be adjacent (though they can be), and there must be at least 3 and at most 6 monads between them.	[Clause [Word] .. BETWEEN 2 AND 5 [Word] ]

#### 4.3.4 Feature-restrictions

##### Basic feature-restrictions

Construction	Meaning	Example
[A myfeature = val]	<p>Feature-equality: A's feature "myfeature" must have value "val".</p> <p>Other comparison-operators include:</p> <ul style="list-style-type: none"> <li>• "&lt;&gt;": inequality (different from)</li> <li>• "&lt;": less than</li> <li>• "&gt;": greater than</li> <li>• "&lt;=": less than or equal to</li> <li>• "&gt;=": greater than or equal to</li> </ul>	[Word lemma="see"]
[A myfeature IN (value-list)]	<p>Value-list: A.myfeature must be an enumeration, and value-list must be a comma-separated list of enumeration constants in parentheses. The meaning is as if an OR had been placed between individual equality (=) comparisons between the feature and the members of the list.</p>	[Word pos IN (article,noun, conjunction, adjective)]
[A myfeature ~ "regex"]	<p>Regular expression: A.myfeature is matched via the regular expression "regex". The regular expressions are compatible with Perl 5. Can only be used with string-features.</p>	[Word lemma ~ "A(b a e)*"]
[A myfeature !~ "regex"]	<p>Negated regular expression: Matches those objects for which the feature in question does NOT match the regular expression. Can only be used with string-features.</p>	[Word surface !~ "se(a e)"]

### Boolean combinations of feature-restrictions

[A feature1 = value1 AND feature2 = value2 ]	Conjunction: Both feature-comparisons must be true at the same time for the object to match.	[Word lemma="see" AND tense=past]
[A feature1 = value1 OR feature2 = value2 ]	Disjunction: If either of the feature-comparisons evaluates to true, then the object matches.	[Phrase phrase_type = NP OR phrase_type=PP ]
[A NOT feature1 = value1]	Negation: The feature-comparison must not be true.	[Word NOT pos=verb]
[A (feature1 = value1 OR feature2 = value2) AND feature3 = value3)	Grouping: Parentheses can be used to group feature-comparisons.	[Phrase phrase_type = VP AND (function = Predicate OR function = PredCmpl) ]

#### 4.3.5 First/last

An object can be first, last, or first and last in its context.

Construction	Meaning	Example
[B [A first] ]	The A object must be first in the context of the B object.	[Phrase [Word first] [Word] ]
[B [A last] ]	The A object must be last in the context of the B object.	[Phrase [Word] .. [Word last] ]
[B [A first and last] ]	The A object must be both first and last in the context of the B object.	[Phrase [Word first and last] ]

#### 4.3.6 Object references

You can give an object a name with the "AS" keyword and then refer to that object later in the query with the "dot notation".

Construction	Meaning	Example
[A AS a1 [B feature_on_B = a1.feature_on_A] ]	B's feature_on_B feature must be the same as the feature_on_A feature on the A object.	[Phrase AS p1 // The phrase must // be the immediate // ancestor of the // word [Word parent = p1.self] ]

#### 4.3.7 NOTEXIST and Kleene Star

The NOTEXIST keyword tells that an object must not exist at a given point.

The Kleene-Star tells that an object must occur either 0, 1, or more times.

The Kleene-Star with a set of integers tucked behind tells the exact number of times the object may occur.

Currently, the Kleene Star cannot be used on the first object in a context, or the first object after a "...".

Construction	Meaning	Example
[A NOTEXIST [B]]	There must not exist a B inside of A.	[Sentence NOTEXIST [Word surface="saw"] ]
[A [B] [C]* ]	Inside of A, there must be a B, followed by zero or more C's.	[Phrase [Word first pos=preposition] [Word pos IN (article,noun, conjunction)]* ]
[A [B] [C]*{0,1} ]	Inside of A, there must occur a B object, followed by either 0 or 1 C objects. Note how this makes the C object optional.	[Clause [word pos="conjunction"] [word pos="conjunction"] ]*{0,1} ]