

Australian National University  
Department of Computer Science

# PADRE<sup>1</sup>

## User Manual

(describes the MPI Version of **padre**)

David Hawking  
dave@cs.anu.edu.au  
Peter Bailey  
peterb@cs.anu.edu.au

August 28, 1996

<sup>1</sup>Development carried out under the ANU - Fujitsu CAP Project and the ACSys Cooperative Research Centre

# Contents

<b>1</b>	<b>INTRODUCTION</b>	<b>4</b>
1.1	A Choice Of Pattern Matching Methods . . . . .	4
1.1.1	Full Text Scanning Method . . . . .	4
1.1.2	Memory Resident Index Method . . . . .	4
1.1.3	Super Dictionary Method . . . . .	4
1.2	Combining the Benefits of FTS and SD Methods . . . . .	4
1.3	What Is A Document? . . . . .	4
1.4	What Is A Match Set? . . . . .	5
1.5	Document Sets . . . . .	5
1.6	The Match Set Stack . . . . .	5
<b>2</b>	<b>ELEMENTARY USE OF PADRE</b>	<b>5</b>
2.1	Padre Environment . . . . .	5
2.2	Starting <b>padre</b> . . . . .	5
2.3	Command Line Options . . . . .	6
2.4	Quitting <b>padre</b> . . . . .	6
2.5	Loading Data for FTS and MRI Methods . . . . .	6
2.5.1	Loading a Subset of Documents . . . . .	6
2.5.2	A Demonstration Document Collection . . . . .	7
2.5.3	Multiple Text Bases . . . . .	7
2.6	Basic Searching in FTS Method . . . . .	7
2.7	Searching For Multiple Alternate Strings . . . . .	8
2.8	Case Sensitivity . . . . .	8
2.9	Displaying Results . . . . .	8
<b>3</b>	<b>CALCULATING RELEVANCE OF DOCUMENTS</b>	<b>9</b>
3.1	Frequency Based Measures . . . . .	9
3.1.1	Using Alternative Relevance Formulae . . . . .	9
3.2	Distance Based Measures . . . . .	10
3.2.1	Znear Command (** Marked for Obsolescence **) . . . . .	10
3.3	Alternative Distance Based Formulae . . . . .	10
3.3.1	Span Command . . . . .	11
3.4	Forcing Documents to be Considered Relevant or Irrelevant . . . . .	12
3.5	Gory Details Of Ranking Relevance . . . . .	13
<b>4</b>	<b>ADVANCED USE OF PADRE</b>	<b>13</b>
4.1	Changing Padre Behaviour . . . . .	13
4.2	The Reset and Topic Commands . . . . .	13
4.3	Regular Expression Matching - FTS Method Only . . . . .	14
4.4	Proximity Operators . . . . .	14
4.4.1	Followed By . . . . .	15
4.4.2	Not Followed By . . . . .	15
4.4.3	Preceded By - (** NOT IMPLEMENTED **) . . . . .	15
4.4.4	Not Preceded By . . . . .	15
4.4.5	Near . . . . .	15
4.4.6	Not Near . . . . .	16
4.5	Restricting Searches To Components Of Documents . . . . .	16

4.5.1	Within . . . . .	16
4.5.2	Including . . . . .	16
4.6	Operations on Match Sets . . . . .	17
4.6.1	Set Operators . . . . .	17
4.6.2	Infix Set Expressions . . . . .	17
4.6.3	Set Names . . . . .	17
4.6.4	Union . . . . .	18
4.6.5	Intersection . . . . .	18
4.6.6	Difference . . . . .	18
4.6.7	Converting Match Sets . . . . .	19
4.6.8	Loading and Saving Match Sets . . . . .	19
4.7	Creating and Working With Document Sets . . . . .	19
4.7.1	Converting a Matchset into a Document Set . . . . .	19
4.7.2	Other Ways of Using the Negation Operator . . . . .	20
4.7.3	The <code>makedocset</code> Command . . . . .	20
4.7.4	Making a Doc Set From a List of Document Names . . . . .	21
4.7.5	Using a Doc Set to Filter Matches . . . . .	21
4.7.6	Loading Documents Indicated By a Doc Set . . . . .	21
4.7.7	Listing Titles of Documents In a Docset . . . . .	22
4.7.8	Making Wordlists . . . . .	22
4.8	Displaying Lexicographic Context . . . . .	22
4.8.1	Print . . . . .	22
4.8.2	Sample . . . . .	23
4.8.3	Save . . . . .	23
4.8.4	Changing the Form of the Output - Print, Sample and Save . . . . .	23
4.8.5	Word-based Context . . . . .	23
<b>5</b>	<b>TERM CO-OCCURRENCE CAPABILITIES</b>	<b>24</b>
5.1	Term-Term Implications (** Under Construction **) . . . . .	24
<b>6</b>	<b>FACILITIES FOR TEXT BASE ADMINISTRATORS</b>	<b>25</b>
6.1	Enabling More Efficient Loading . . . . .	25
6.2	Removing entries from a text base . . . . .	25
6.3	Removing components from a collection . . . . .	25
6.4	Combining Text Bases . . . . .	26
6.5	Dumping A Text Base . . . . .	26
<b>7</b>	<b>DBMerging Simulation</b>	<b>26</b>
7.1	Lightweight Probes . . . . .	26
7.2	Setting Cell Masks . . . . .	26
<b>8</b>	<b>IMPROVING EFFICIENCY</b>	<b>27</b>
8.1	Load Balance - FTS Method only . . . . .	27
<b>9</b>	<b>MEMORY RESIDENT INDEX METHOD</b>	<b>27</b>

<b>10 SUPER DICTIONARY METHOD</b>	<b>27</b>
10.1 Loading a super-dictionary . . . . .	27
10.2 Searching with a super-dictionary . . . . .	27
10.3 Restrictions on Use of Super Dictionaries . . . . .	28
<b>A SYNTAX FOR REGULAR EXPRESSIONS</b>	<b>30</b>
<b>B SUMMARY OF COMMANDS</b>	<b>32</b>
<b>C SUMMARY OF VARIABLES</b>	<b>35</b>

# 1 INTRODUCTION

The program **padre** (PARallel Document Retrieval Engine) is a further evolution of the **paddy** and **fttr** programs. Much of the functionality of **paddy** was a parallel emulation of similar serial commands in the **pat** (version 3.3) program from the University of Waterloo. **Padre** is currently implemented only for the Fujitsu AP1000. Various papers listed in the **padre** WEB page outline the theory behind PADRE and describe data structures, internal algorithms and the like.

**Padre** is essentially a parallel pattern matching engine which can work over a large collection of documents. Once the pattern-matching machinery has found a set of matchpoints in the textbase, the **padre** user can display the context in which each (or some) of the matches occurred (for linguistic or lexicographic purposes) or use relevance measures calculated by **padre** as the basis of identifying or retrieving the most relevant documents to their research topic.

## 1.1 A Choice Of Pattern Matching Methods

**Padre** provides three different methods for locating matchpoints in the text. Each has advantages and disadvantages.

### 1.1.1 Full Text Scanning Method

This method loads entire text bases into RAM memory of the cells and performs complete scans through them in order to locate matchpoints. FTS is slowest at locating simple patterns but supports regular expression matching and `{wsmode any}` which the other methods don't. The size of textbases is limited by available memory.

### 1.1.2 Memory Resident Index Method

MRI method is very fast because it keeps a wordstart index in memory as well as the raw text. Doesn't support `regexp` or `{wsmode any}`. Even more tightly limited by available memory because the word start index is typically two thirds of the size of the raw text.

### 1.1.3 Super Dictionary Method

SD method is capable of handling much larger document collections than the other two and offers the same capabilities as (or better than) MRI method. The textbase size is limited by available disk space more than memory. Doesn't support `regexp` or `{wsmode any}`.

## 1.2 Combining the Benefits of FTS and SD Methods

Work is going on in 1996 to try to realise the benefits of both these methods at the same time when queries can be structured in two phases - a recall phase based on superdictionary method and a ranking phase. Documents identified in the recall phase are loaded into memory as a pseudo collection and ranked using the FTS method.

## 1.3 What Is A Document?

**Padre** must be given a character string which marks the beginning of every document. The piece of text starting with the first character of the marker and ending immediately before the first character of the next one is defined to be a document. Documents are not split across AP1000 cells.

For retrieval purposes, documents should not be excessively long. If the collection includes omnibus volumes such as dictionaries or encyclopaedias, each entry should be marked as a separate document. However, for some purposes, it would be preferable to treat the whole omnibus as a single document. Future versions of **padre** are likely to adopt a more flexible definition of what constitutes a document.

## 1.4 What Is A Match Set?

Internally, the result of most **padre** operations is an ordered set of pointers to characters in the text collection. This is called a match set. If a search is made for the pattern "cat" the result will be a set of pointers to every letter "c" which is followed by "at". The first pointer will reference the first occurrence of "cat", the second pointer will reference the second occurrence and so on.

**Padre** allows match sets to be named and referenced again later in the same query.

## 1.5 Document Sets

**Padre** includes a number of operators which create special match sets in which the matchpoints are all pointers to the very first character of each of a subset (possibly the complete set) of documents.

## 1.6 The Match Set Stack

As match sets are computed by search operations they are stored on an upwards-growing stack. Operators which operate on match sets are usually defined to replace the top match set (or the top  $n$  match sets) with a single result set.

# 2 ELEMENTARY USE OF PADRE

## 2.1 Padre Environment

Before using **padre** you should set the environment variable `PADREHOME` to the appropriate directory for your system. This can be done in your `.cshrc` file.

## 2.2 Starting padre

To start **padre**, type:

```
cd $PADREHOME/bin
mpirun -inplace 10000000 padre ftr_cell
```

You should now see the **padre** command prompt:

```
>>
```

This prompt signifies **padre**'s readiness to accept a command.

To operate **padre** in non-interactive mode, supply input and output files via Unix redirection. ie.

```
mpirun padre ftr_cell <cmds >cmds.log
```

## 2.3 Command Line Options

- trec - format the lists of retrieved documents to suit the TREC competition.
- par1 - format the lists of retrieved documents to suit the WAIS server.
- terse - reduce the volume of informational about numbers of matches, times etc.

## 2.4 Quitting padre

To quit **padre** type the following command :

```
>> quit
```

## 2.5 Loading Data for FTS and MRI Methods

Before information can be retrieved in these methods, the document collection must be loaded into memory. The AP1000 consists of a number of cells, ranging from 16 to 1024, each with its own memory. To load data into the AP1000, **padre** loads a portion of the data into each cell.

The document collection may be stored either on host disks (either compressed or uncompressed) or on DDV option disks (using the HiDIOS filesystem). There are a number of different load commands:

```
>> load "filestem" - uncompressed host file
>> cload "file or directory-name" - compressed host files
>> doload "filestem" - DDV option filesystem
```

When the load command requires a *filestem*, **padre** expects to find the textbase in a file called *filestem.tb*. In the case of the **load** command it also expects to find a companion description file *filestem.info*. At the moment, the only information in the *.info* file is the string which marks the beginning of a document. The same information is held in a file called *.info* in the directory containing compressed text files and in headers stored in the files on DDV disks.

The **cload** command recursively scans the directory tree starting at the specified directory and loads as text data every ordinary file which is not called *.info* or *README*. If the number of files to load is not an even multiple of the number of AP1000 cells, the **loadbalance** command may be used to distribute the data more evenly.

A variant of the **cload** command takes a file rather than a directory as its argument. The first line of the named file is interpreted as the start of document marker (*¡DOC¡* in the case of TREC data). The first “word” on each subsequent line is treated as the name of a compressed file to load.

### 2.5.1 Loading a Subset of Documents

The **cload** command now takes an optional numeric parameter *n*:

```
>> cload "directory-name" "n"
```

which allows only a sample of the data to be loaded. Only every *n*th file encountered in the directory scan is loaded. This can be useful in performing term implications or collection categorisations when time or memory limits are imposed.

### 2.5.2 A Demonstration Document Collection

A sample text base consisting of a directory of compressed Usenet news items is sometimes supplied with **padre**. It is usually called `/text/news`. Each news article starts with the SGML tag `<item>`. Accordingly, the file `/text/news/.info` contains a single line with the string `<item>`.

Thus, to load the sample text base, use

```
>> cload "/text/news"
```

### 2.5.3 Multiple Text Bases

In the FTS method, **Padre** makes it possible to have several text bases in memory at once, by allowing multiple load, cloud or doload commands. To switch between the text bases in memory, change the variable *base*. For example:

```
>> {base 3}           - switch to the third base in memory
```

Note that the first textbase loaded is number 1.

## 2.6 Basic Searching in FTS Method

The simplest **padre** searches specify a literal pattern which is matched exhaustively against the entire text base using a modified Boyer-Moore-Gosper algorithm. The pattern can include spaces, where a space matches a space or any single non-alphanumeric. If the search pattern contains spaces, the entire pattern must be enclosed by double quotes. By default, matches are constrained to lie at word starts but this can be changed by changing the value of the *wsmode* variable (see example below). Some example searches are:

```
>> "cat"              - matches any word starting with "cat" eg. "catch"
>> "cat "             - matches only the word "cat" (or "CAT", "Cat", etc.)
>> {wsmode any}
>> "cat"              - search for every occurrence of "cat" regardless
                        of word boundaries, eg. "educate"
>> "ize "             - matches any word ending in "ize", eg. "sanitize"
>> "cats and dogs "   - matches the three words separated by single
                        spaces or single punctuation marks.
```

Though they may often be omitted, it is a good idea to enclose each search string in quotes in case the string happens to be the name of a **padre** command. For example :

```
>> "load"             - search for all occurrences of the string load
```

After a search command has been completed, **padre** displays the number of matchpoints found and the time taken to complete the search. Times are measured on the AP1000 host machine (front-end).

```
>> textual
   9 matches.
Search time: elapsed =    0.03 sec., cpu =  0.033 sec
```

## 2.7 Searching For Multiple Alternate Strings

It is often necessary to search for occurrences of any one of a list of alternates. For example, references to any one of the countries in Europe. The current version of `padre` includes two different commands (based on different algorithms) for performing this task in a single scan through the data. These are the `anyof` and `bmg2` commands. The following example shows two commands which should produce identical results:

```
>> anyof "france |germany|england |switzerland |italy spain"
>> bmg2 "'france |germany|england |switzerland |italy spain"
```

In future versions of `padre`, the `anyof` and `bmg2` commands are likely to disappear and be merged with the simple string-searching case:

```
>> "france |germany|england |switzerland |italy spain"
```

## 2.8 Case Sensitivity

By default, `padre` searches are not case-sensitive. Ie. "a" matches both "a" and "A". To make searches case sensitive, specify:

```
{casesensitive 1}
```

## 2.9 Displaying Results

The command `top n` lists the  $n$  documents judged most relevant to the topic so far (in descending order of estimated relevance.) If the documents included a `<DOCNO>` field, the contents of the field are displayed as a title.

If `padre` is operating in *parliamentary* mode (`-parl` command line switch or mode `parl`), the `top` command lists each document with an identifier of the form [*cell num, collection num, doc num*]<sup>1</sup>. This identifier can be supplied as an argument to the `getdoc` command in order to display a particular document.

```
>> top 3
[35, 2, 54] WSJ870227-0021
[3, 1, 528] WSJ881103-0102
[55, 2, 120] WSJ861222-0120
>> getdoc "[35, 2, 54]"
<DOC>
<DOCNO> WSJ870227-0021 </DOCNO>
<HL> Personal Taxes (A Special Report): Work Sheets
By Gay Sands Miller</HL>
...
```

The command `retrieve n` retrieves the text of the  $n$  documents judged most relevant to the topic so far into a file in the current directory called `reldocs.topicname`. Extremely long documents (over half a megabyte) are truncated.<sup>2</sup>

---

<sup>1</sup>Check that this really happens!

<sup>2</sup>This restriction should be removed. It has caused Paul problems.

## 3 CALCULATING RELEVANCE OF DOCUMENTS

**Padre** offers two completely different ways of scoring the relevance of documents:

**frequency-based** A family of different formulae which use frequency of occurrence of query terms in a particular document and in the collection as a whole to derive relevance scores. Sometimes the length of the document is also taken into account.

**distance-based** A new family of formulae which ignore collection statistics and document lengths completely. Instead scores are computed according to span lengths between groups of terms. Of course there is a frequency component in that each span within a document adds to its score.

The **padre** user must indicate the start of a new research topic by issuing the topic command `topic topicname`. This command resets the cumulative relevance metric for each document in the collection.

If desired, the cumulative relevance metrics for each document may be cleared using the `reset weights` command. This command has no effect on the flags which indicate whether the document is mandatorily included or excluded. This capability allows the searcher to define a subset of documents for further consideration without the weights calculated in defining the subset counting in the final relevance assessment. (See `include` and `exclude` commands below.

### 3.1 Frequency Based Measures

Each time a pattern matching, proximity or set operation is executed, the relevance of each document containing matchpoints is updated according to how many matchpoints it contains and how many matchpoints there are altogether. A bias against long documents is also applied.

Particular search terms may be assigned an importance weighting using the `weight` variable. By default, the weight is 5. In the following example, the presence of Clinton or Yeltsin by themselves is considered totally irrelevant but the presence of the two of them in proximity is considered highly important.

```
>> {weight 0}
>> Clinton
>> Yeltsin
>> {weight 10}
>> near 2
```

Negative weights may be used to downgrade the assessed relevance of documents containing particular patterns.

#### 3.1.1 Using Alternative Relevance Formulae

**Padre** currently supports several alternative formulae for calculating a document's raw weight with respect to each **padre** operation. The user can select which one is used by changing the `relmode` variable. Note that raw weights are multiplied by the manually assigned weight before being accumulated.

`relmode weighted` - The default. Raw weight is calculated as the number of matches in a document divided by log to the base 10 of the product of the document length and the total number of matches.

`relmode boolean` - Raw weight is set to 1.0 if the document contains one or more matches, to 0.0 if it does not.

`relmode nocf` - As for `weighted` except that the total number of matches term (the *collection frequency* is dropped).

`relmode prologs` - As for `weighted` except using the product of the logs rather than the log of the product.

`relmode squirrel` - As for `weighted` except that `sqrt` rather than `log` is employed.

`relmode zmode` - Relevance is NOT accrued by normal searches at all . This allows conventional ranking to be turned off while the `znear` (q.v.) command is being used.

`relmode count` - As for `boolean` but the raw weight is set to the number of matches in the document.

## 3.2 Distance Based Measures

Calculating relevance on the basis of distance is best illustrated by example. Imagine that we are seeking documents relevant to “the economic impact of recycling tyres”. We may believe that relevant documents will include references to the three concepts represented by “economic impact”, “recycling” and “tyres”.

Imagine that a document contains one instance of each concept. The closer together are the instances, the higher will be the score, reflecting the increase in probability that the occurrences are related.

Now imagine a document apparently containing only two of the concepts, say “recycling” and “tyres”. Such a document should score less than a document containing all three concepts but it should be given a non-zero score on the basis that the “economic impact” concept may be present but in an unrecognised form such as “increased profits”.

In some cases, relevant documents may be identified by singleton terms or phrases. For example, “self hypnosis”. These can be regarded as saturated complete spans of length 1.

### 3.2.1 Znear Command (\*\* Marked for Obsolescence \*\*)

The `znear` command operates in the same way and has the same function as the `near` but updates relevance in a way which takes into account the minimum span (in words) between the hits forming a proximity set. It scores nothing for partial spans.

## 3.3 Alternative Distance Based Formulae

For each span, a raw-weight is calculated as a fraction consisting of a numerator divided by a denominator. The `zmode` variable is taken as a two digit decimal number in which the tens digit controls what is used as the numerator: (0 implies unity, 1 implies the average of the number of proximal hits for each participating term, 2 implies the sum of the proximal hits.) and the units digit controls the denominator: (0 implies the span in words, 1 implies the square root of the span and 2 implies the maximum of 5 and the span). The value of 33 gives the formula used by the University of Waterloo in TREC4. The value of 01 is the formula found best by ANU in TREC4.

### 3.3.1 Span Command

The `span` command differs from `znear` in that:

1. its parameters are specified on the command line
2. it produces no result set
3. it assigns scaled down scores for partial spans

There are three basic varieties of `span` command which differ in the way partial spans are handled.

```
>> span all num_sets weight prox_limit min_num_sets
>> span leading num_sets weight prox_limit min_num_sets
>> span key num_sets weight prox_limit min_num_sets num_key_sets
```

All three variants require the specification of:

1. the number of sets over which relevance is to be computed,
2. the multiplicative factor (weight) used to scale resulting span scores,
3. the proximity range in characters to which spans are limited, and
4. the minimum number of terms (one from each set) required to form a span.

The `all` variant computes span scores for all possible combinations of `num_sets` sets taken `min_num_sets` or more at a time. It may be useful when all the concepts in a concept intersection are equally meaningful. If sets 1, 2, 3, 4 are on top of the matchset stack when the following example command is processed,

```
>> span all 4 1000 1000 3
```

the following combinations of sets will be scored:

1, 2, 3, 4, 1, 2, 3, 1, 2, 4, 1, 3, 4, 2, 3, 4

The `leading` variant computes span scores only for sequences of sets starting with the one first placed on the stack. If sets 1, 2, 3, 4 are on top of the matchset stack when the following example command is processed,

```
>> span leading 4 1000 1000 2
```

only the following combinations of sets will be scored:

1, 2, 3, 4, 1, 2, 3, 1, 2

The `key` variant computes span scores only for the combinations of sets which would be used in the `all` variant minus those which do not include the key set or sets. The key set or sets must be placed on the stack before the others. If sets 1, 2, 3, 4 are on top of the matchset stack when the following example command is processed,

```
>> span key 4 1000 1000 3 1
```

only the following combinations of sets will be scored:

1, 2, 3, 4, 1, 2, 3, 1, 2, 4, 1, 3, 4

A `key` variant might be appropriate in a case where several of the concepts being intersected were quite broad and one or two were very specific. For example, in retrieving documents relevant to *the economic management of film studios* the concepts *economic* and *management* are very broad whereas relatively few documents deal with film studios.

### 3.4 Forcing Documents to be Considered Relevant or Irrelevant

The `include` and `exclude` commands may be used to over-ride the normal relevance ranking formula. Several modes are supported. They are described by means of examples.

```
>> "computer"  
>> exclude current
```

`Exclude current` will cause documents containing matches in the current match set (in the example all those containing the word `computer`) to be treated as irrelevant.

```
>> "computer"  
>> exclude others
```

`Exclude others` will cause documents NOT containing matches in the current match set (in the example all those NOT containing the word `computer`) to be treated as irrelevant.

```
>> exclude biased
```

`Exclude biased` will cause documents whose accumulated positive relevance derives more than 85% from a single search term to be treated as irrelevant.

```
>> exclude -0.5
```

`Exclude -threshold` will cause documents whose accumulated negative relevance exceeds a threshold value to be treated as irrelevant. (Documents for which there is strong negative evidence.)

```
>> exclude +0.5
```

`Exclude +threshold` will cause documents whose accumulated positive relevance does not exceed a threshold value to be treated as irrelevant. (Documents for which positive evidence is weak.)

```
>> "computer"  
>> include current
```

`Include current` will cause documents containing matches in the current match set (in the example all those containing the word `computer`) to be treated as relevant.

```
>> "computer"  
>> include others
```

`Include others` will cause documents NOT containing matches in the current match set (in the example all those NOT containing the word `computer`) to be treated as relevant.

```
>> include -0.5
```

`Include -threshold` will cause documents whose accumulated negative relevance does not exceed a threshold value to be treated as relevant. (Documents for which negative evidence is weak.)

```
>> include +0.5
```

`Include +threshold` will cause documents whose accumulated positive relevance exceeds a threshold value to be treated as relevant. (Documents for which positive evidence is strong.)

### 3.5 Gory Details Of Ranking Relevance

Use of the `include` and `exclude` commands may result in apparent conflict or ambiguity in ranking a particular document. **Padre** resolves such problems by dividing documents into three categories:

1. Documents mandatorily included and NOT excluded.
2. Documents both included AND excluded OR neither.
3. Documents mandatorily excluded.

Within categories, documents are ranked according to the sum of their accumulated positive and negative relevance. Documents in the first category are eligible to be returned even if this sum is very small or negative. Documents in the second category are only eligible to be returned if the sum exceeds zero. Documents in the third category are ineligible to be returned.

Category 1 documents are processed first, then category 2. If the number of documents requested is less than the number of documents in category 1, the lowest ranked documents will not be returned even though their inclusion is said to be mandatory.

## 4 ADVANCED USE OF PADRE

### 4.1 Changing Padre Behaviour

As has been seen in some examples, **padre** behaviour is controlled by the values of certain variables. These variables are listed in Appendix C. Variable assignments are enclosed in braces: `{variable value}`. For example:

```
{wsmode any}
```

The role of particular variables is described in the context of the functions they modify.

### 4.2 The Reset and Topic Commands

The `weights` modifier to the `reset` command has been mentioned earlier. Here is a full list of modifiers:

**all** Clear all match sets, document accumulators, component tables and pseudo collections.

In the case where a `loaddocs` command has been used to load a subset of documents as a pseudo textbase, the `reset all` command, clears everything related to the pseudo textbase and reverts to SDmode with the previously active superdictionary.

**bits** clear all bits (include, exclude and future) bits associated with each document.

**start** Reverts **padre** to a state as close as possible to the state at the beginning of the current execution.

**vol** clear all volatile sets from the stack.

**weights** clear the relevance accumulators associated with each document.

The `topic` command which requires a topic name as its argument has two effects. It is used to label the lists of document ids produced in `-trec` mode and it also causes a reset which clears match sets, document accumulators and component tables and resets most variables which affect relevance (proximity, weight, casesensitive, WSmode) to their standard values. Other variables are not altered.

### 4.3 Regular Expression Matching - FTS Method Only

**Padre** provides the ability to search for regular expressions. Regular expressions allow more general patterns to be specified than with the other search methods. The regular expression search routine comes from the gnu “Extended Regular Expression Matching and Search Library”. To search for a regular expression, type the **padre** command **regexp** followed by the regular expression to search for. Some examples are :

```
>> regexp "\<[a-z]*ize\>"
```

The above regular expression searches for words ending in **ize**, for example **size** and **privatize**. To do this, the pattern matches the start of a word (**\<**), followed by zero or more letters (**[a-z]\***), followed by **ize**, followed by an end of word marker (**\>**).

```
>> regexp "\<[a-z]*consider[a-z]*\>"
```

The above regular expression searches for words with **consider** in the middle, for example **considering** and **inconsiderate**. To do this, the pattern matches the start of a word (**\<**), followed by zero or more letters (**[a-z]\***), followed by **consider**, followed by zero or more letters (**[a-z]\***), followed by an end of word marker (**\>**).

```
>> regexp "\<[a-hj-z]*i[a-hj-z]*i[a-hj-z]*\>"
```

The above regular expression searches for words with exactly two **i**'s, for example **participation** and **building**. To do this, the pattern matches the start of a word (**\<**), followed by zero or more letters that are not **i**'s (**[a-hj-z]\***), followed by an **i**, followed by zero or more letters that are not **i**'s (**[a-hj-z]\***), followed by an **i**, followed by zero or more letters (**[a-hj-z]\***), followed by the end of word marker (**\>**).

See appendix A for syntax of regular expressions.

### 4.4 Proximity Operators

Proximity operators produce a result match set from an arbitrary number  $n$  of previously computed match sets. The  $n$  operands are removed from the stack and replaced by the result set.

Proximity is defined in terms of the number of characters separating two matchpoints. A member is added to the result set if and only if an  $n$ -tuple satisfying the proximity relation can be formed with one element from each of the component sets.

The range for the proximity searches is defined by the proximity variable. The default proximity is 100 characters. To change the proximity to a different value, say 140 characters, use the command :

```
>> {proximity 140}
```

The proximity is measured from the first character of the first match to the first character of the second match. Proximity is not permitted to extend beyond the boundaries of a document. In the following examples the prevailing proximity is represented as  $p$ .

#### 4.4.1 Followed By

The result of a `fbym` operation consists of all members of the first set followed within proximity by a member of each of the other sets in order. The following example searches for occurrences of `cat` that are followed by occurrences of `dog` followed by `aardvark`, all within  $p$  characters.

```
>> "cat "  
>> "dog "  
>> "aardvark "  
>> fbym 3
```

#### 4.4.2 Not Followed By

The result of a `not fbym` operation consists of all members of the first set which are not followed within proximity by a member of the second set. The following example searches for occurrences of `cat` that are not followed by an occurrence of `dog` within  $p$  characters.

```
>> "cat"  
>> "dog"  
>> not fbym
```

#### 4.4.3 Preceded By - (\*\* NOT IMPLEMENTED \*\*)

The result of a `pbym` operation consists of all members of the first set preceded within proximity by a member of each of the other sets in order. The following example searches for occurrences of `cat` that are preceded by occurrences of `dog` preceded by `aardvark`, all within  $p$  characters.

```
>> "cat "  
>> "dog "  
>> "aardvark "  
>> pbym 3
```

#### 4.4.4 Not Preceded By

The result of a `not pbym` operation consists of all members of the first set which are not preceded within proximity by a member of the second set. The following example searches for occurrences of `cat` that are not preceded by an occurrence of `dog` within  $p$  characters.

```
>> "cat"  
>> "dog"  
>> not pbym
```

#### 4.4.5 Near

The result of a `near` operation consists of all members of any of the component sets followed within proximity by a member of each of the other sets, not necessarily in order. The following example searches for occurrences of `cat`, `dog` and `marmot` within  $p$  characters of each other. The result set is sorted into order.

```
>> "cat"
>> "dog"
>> "marmot"
>> near 3
>>
```

#### 4.4.6 Not Near

The result of a `not near` operation consists of all members of the first component sets which are not followed or preceded within proximity by a member of the second set. The following example searches for occurrences of `cat` for which there is no occurrence of `dog` within  $p$  characters.

```
>> "cat"
>> "dog"
>> not near
>>
```

### 4.5 Restricting Searches To Components Of Documents

If parts of a document such as title, authorname, date etc. are delineated by start and end markers, they can be defined as **padre** components. Subsequent searches can be constrained to look only within particular named components. In the current version of **padre** only literal searches can be restricted in this way, but this restriction may eventually be removed.

To define a chapter component with start marker `<ch>` and end marker `</ch>` the following command may be used :

```
>> chapter = component <ch>..</ch>
```

In future versions of **padre**, regular expressions as start and end markers may be supported. Indeed the `within` and `including` commands are ripe for redevelopment, so that more sophisticated things can be done<sup>3</sup>

#### 4.5.1 Within

The `within` command searches for text that is contained in a certain type of component. In the following example, the result set includes only the occurrences of `rubbish` which occur between the above markers.

```
>> "rubbish" within component chapter
```

#### 4.5.2 Including

The `including` command searches for components of text that contain the specified pattern. The result set produced by the following command includes pointers to all chapter components containing the string `rubbish`.

```
>> component chapter including "rubbish"
```

---

<sup>3</sup>Change needed!

## 4.6 Operations on Match Sets

As stated above, the result of a search is called a match set, or more simply, a set, in which each element is a pointer to the first character of a match in the text. The results of two searches can be used to obtain a third set using one of the set operators. Each of the items in a set is a match point, so the set operators do their comparisons on match points. To perform a set operation on two sets, separate the commands to find the two sets with the desired set operator.

### 4.6.1 Set Operators

Union, Intersection and Difference operators are provided in both infix and “postfix” forms. Infix operators are used within single line expressions and postfix operators replace the top  $n$  sets on the stack with the result of applying the operator. A Negation operator is also provided in both forms but converts its operand to a docset if it is not already one.

### 4.6.2 Infix Set Expressions

Multiple set operations are permitted within a single line. At present, all operators are considered to have the same precedence and operators are evaluated left-to-right.

```
>> "dog " + "cat " + "cow " + "horse "  
>> "communist "  
>> ~                               - result is set of all docs not  
                                   - containing "communist "
```

### 4.6.3 Set Names

It is also possible to name a set, so that it can be used in a later set operation. A set is named with the command :

```
>> {setname name}    - where name is some alphanumeric string, where  
                     the first character must be a letter, not a digit
```

The identifier *name* now refers to the most recent match set computed by **padre**. If *name* has already been used in a previous setname command, then the old set value will be overwritten with the new one. It is of course an error to perform a setname operation before any entry sets have been computed.

Having named a set, it can be used within a set operation as follows:

```
>> colour  
>> blue  
>> fby 2  
  
>> {setname bluecol}  
  
>> colour  
>> orange  
>> fby 2  
>> {setname orangecol}
```

```
>> {bluecol} ^ {orangecol}
```

```
1 matches in result set.
```

```
Time for set operation: elapsed = 0.05 sec., cpu = 0.000 sec
```

Invoking the name of a set causes a copy of it to be put on top of the matchset stack. Setnames evaporate when a `topic` command is issued.

#### 4.6.4 Union

The union operator builds a third set which contains every element of the first and second set. The union operator creates an ordered set with no duplicate items. An example of using the union operator is :

```
>> cat + dog          - Find all patterns starting with cat or dog
```

Alternatively,

```
>> cat
>> dog
>> union 2
```

#### 4.6.5 Intersection

The intersection operator builds a third set that contains only those elements that occur in both the first and second set. An example of the infix intersection operator has already been given. In postfix form:

```
>> {bluecol}
>> {orangecol}
>> intersection 2
```

#### 4.6.6 Difference

The difference operator builds a third set that contains all elements that occur in the first set but do not occur in the second set. An example of the difference operator is :

```
>> compute - computer    - Find all matches starting with compute, but not
                           starting with computer.
```

Or, equivalently:

```
>> compute
>> computer
>> diff 2
```

### 4.6.7 Converting Match Sets

*Not available in SD mode.*

**Padre** provides an operator `matchpreceding` to replace each matchpoint in a match set with the immediately preceding occurrence of some pattern (regular expression). Pre-defined patterns are available to match sentence starts, paragraph starts or document starts, but the user may supply their own. Using `matchpreceding` and the set operators allows one to find sentences containing both of two terms. This functionality overlaps to some extent with the component search operators and with the document set operators.

`matchpreceding` takes one argument, either `sent`, `para`, `doc` or a regular expression. The following example finds all sentences containing "dog " or "dogs" and "fleas ".

```
>> bmg2 "dog |dogs "  
>> matchpreceding sent  
>> "fleas "  
>> matchpreceding sent  
>> intersection 2
```

### 4.6.8 Loading and Saving Match Sets

Sometimes the usefulness of a set of matchpoints (such as the set of occurrences of all terms defining a U.S. context) outlives a particular topic or even a particular run of PADRE. Such a matchset may be saved on option disk using the `savematchset` command. The filename is assumed to have a `.mset` suffix.

```
>> use matchsetfile filestem  
>> savematchset name  
>> loadmatchset name
```

The `use matchsetfile` command creates a new option disk file and writes a record identifying the superdictionary or textbase in use plus some sort of CRC value so that attempts to load irrelevant matchsets can be detected and avoided. Subsequent calls to `savematchset` write a record recording `name` followed by the current set of match pointers. `Name` can be anything specified by the user.

`Loadmatchset` retrieves the first matchset corresponding to `name` from the file and loads its matchpoints as a new set on the top of the stack.

## 4.7 Creating and Working With Document Sets

A document set is a special case of a match set. Consequently, set and proximity operators may be used with document sets. Some additional commands are needed to create and manipulate document sets, however.

### 4.7.1 Converting a Matchset into a Document Set

The `matchdocstarts` command converts the topmost match set on the stack into an ordered set of pointers to the starts of all documents which contained matches in the last set.

The negation command `not` converts the topmost match set on the stack into an ordered set of pointers to the starts of all documents which did not contain matches in the last set. The use of both these commands is illustrated here:

DOCUMENTS:

Total=21705, min. no. per cell=122(88), max no. per cell=195(34)

6.29 >> "criminal "

763 matches.

Number of docs including matches: 451

14.95 >> matchdocstarts

312 duplicates eliminated

451 matches.

22.98 >> ~

21254 matches.

#### 4.7.2 Other Ways of Using the Negation Operator

In the example which follows, the negation operator is used as a unary operator within an infix set expression.

```
>> "criminal "  
>> {setname crimset}  
>> ~ crimset + ~ "businessman"
```

#### 4.7.3 The `makedocset` Command

This command provides various ways of making a document set. At present the only forms implemented are:

- `makedocset n` - Make a document set of the  $n$  most relevant documents.
- `makedocset mn` - Make a document set of the  $n$  most relevant documents, but exclude documents with lower rank (higher relevance) than  $m$ . Documents are ranked from zero (most relevant).

Note that `makedocset n` is equivalent to `makedocset 0n` and that `makedocset 23` will return only the document of rank 2.

It is planned to eventually support the following additional modifiers to the `makedocset` command:

- `relevant`,
- `included`,
- `excluded`, and
- `current`.

#### 4.7.4 Making a Doc Set From a List of Document Names

The following commands may be used to create a set of pointers to documents whose names match those in a list supplied either in the command line or in a host file.

```
finddocs "doc1" "doc2" ...
findfiledocs "hostfile"
```

For example:

```
finddocs WSJ910603-0117 CR93H-10986 AP880616-0185 FT941-16483
```

#### 4.7.5 Using a Doc Set to Filter Matches

Sometimes one is interested only in matches found within certain documents. The `withindocset` command removes from a matchset all those matchpoints which lie outside the documents specified in a doc set. In the following:

```
>> matchset
>> docset
>> withindocset
```

the matchset and the docset are replaced by a suitably filtered subset of the matchset. The following example finds all occurrences of "Oswald" which are in documents which also refer to Kennedy. Note that there may be more than one match per document.

```
>> Oswald
>> Kennedy
>> matchdocstarts
>> withindocset
```

#### 4.7.6 Loading Documents Indicated By a Doc Set

*SD mode only.*

The `loaddocs` command creates a new in-memory textbase containing only the documents specified by the document set on top of the stack. This command is the key to the two phase queries mentioned in the introduction. In the following example, the `anyof` defines a set of documents worthy of further analysis. These are loaded into memory and processed. The `regexp` is an illustration of something which can be done in FTS but not in SD mode. The `reset all` command drops back to using the super dictionary.

```
>> usesd SDtwo
>> anyof "economic impact|recycl |tyre"
>> matchdocstarts
>> loaddocset
>> regexp "\$[1-9][0-9]*"
>> ...
>> reset all
```

### 4.7.7 Listing Titles of Documents In a Docset

The `listdocs` command saves a list of the titles of the top matchset to a specified file. For example:

```
>> 'criminal '  
>> makedocset current  
>> listdocs <file>
```

saves all the titles of documents including the word `criminal` in `file`. The `makedocset current` command is necessary to avoid duplicates.

### 4.7.8 Making Wordlists

The `wordlist` command saves in a file a listing of all distinct words which occurred within documents in the topmost docset. With each word is shown the number of documents which contained it. The first line of the file starts with a blank so it will remain first even if the file is sorted and gives a count of the total number of documents in the set.

The `wordlist2` command is exactly analogous to `wordlist` but each word is marked with the number of documents in the first set which contained it and the number of documents in the second set which contained it.

```
>> 'criminal '  
>> matchdocstarts  
>> 'entrepreneur''  
>> matchdocstarts  
>> wordlist2 <file>
```

## 4.8 Displaying Lexicographic Context

In lexicographic or linguistic research, it is useful to display the context in which patterns occur in the text. **Padre** is able to display contexts consisting of a fixed number of characters before the match point, and a fixed number of characters after it. **Padre** shows the address of the match in the form [ *cell-id*, *index within chunk*, *size of that chunk* ] followed by the context of the match. For example:

```
>> "cat"  
>> sample  
[ 0, 355/ 164703] .hest class or category (of roads, academic  
[ 15, 102409/ 161730] .ed.</s2> <cmp>Cat's-eye</cmp> <ge>Brit.</g  
[ 15, 115852/ 161730] ./s2> <ds><drv>catastrophic</drv> <pr><ph>-
```

### 4.8.1 Print

To display the contexts of all matchpoints, use the `print` command. Its syntax is:

```
>> pr
```

## 4.8.2 Sample

When there are a large number of matching entries, the user may wish to see just a few of them. **Padre** provides the `sample` command to do this.

```
>> sample
```

The default size of the sample is 10, but this can be changed by setting the variable `samplesize` to the desired number. For example, to set the sample size to 20, use the following command :

```
>> {samplesize 20}
```

Assuming a numbered ordering of the matches  $1, \dots, n$ , **padre** distributes the sample points evenly across the interval.

## 4.8.3 Save

**Padre** provides the `save` command to save the contexts to a file rather than to the terminal.

```
>> save
```

The contexts will be written to the file `padre.default_savefile`. The file used for saving the results can be changed by putting the desired filename in the variable `savefile`. For example to change the savefile to `newfile`, type the following command :

```
>> {savefile newfile}
```

## 4.8.4 Changing the Form of the Output - Print, Sample and Save

The total number of characters of context printed by **padre** is controlled by the `printcontextlength` variable. The number of characters **padre** will print before the match point is controlled by the `printbefore` variable:

```
>> {printbefore 10} - display 10 characters before the match point
>> {printcontextlength 140} - display 140 characters for the match
```

**padre** also makes it possible to change the position of all the match points in the last set by using the `shift` command. The match point can be moved to the left or to the right. Some examples of using the `shift` command are :

```
>> shift.5 - move the match point to the right 5 characters
>> shift.-3 - move the match point to the left 3 characters
```

## 4.8.5 Word-based Context

The `context` command is similar to `save` except that it works in words rather than characters, does not cross document boundaries and ignores the `printcontextlength` and `printbefore` variables. Its format is

```
context i words beforei j words afterj ifilei
```

```
>> twaddle
>> context 10 10 "~/contexts/twaddle.ctxt"
```

## 5 TERM CO-OCCURRENCE CAPABILITIES

Term co-occurrence information extracted from documents in a collection may be used to derive phrases, associations of terms and possibly synonyms. These relations are potentially valuable in forming queries.

These capabilities are now being built in to **padre**. They are being built on top of information collected in building superdictionaries and indexes.

### 5.1 Term-Term Implications (\*\* Under Construction \*\*)

The various forms of the **implications** command cause a list of implication relationships between terms to be saved in a host file. As an illustration of what this means, if **dog** implies **cat**, then, for the given collection, the probability that documents containing **dog** also contain **cat** exceeds a given threshold.

```
>> implications -list SD_name wordlistfile low_freq high_freq resultfile
>> implications -allwords SD_name low_freq high_freq resultfile
>> implications -saved SD_name matchsetfile low_freq high_freq resultfile
>> implications -phrases SD_name old_resultfile new_resultfile
```

Currently, the **-phrases** form is not implemented and only the **-saved** form has been tested. Use of the others is not advised.

In order to adapt to different collection sizes, the frequency thresholds are interpreted as rates. The low threshold is interpreted as words per billion and the high as words per million. Useful values for low and high are expected to be 150 and 1000 respectively.

The meaning of each of the forms whose required argument lists are shown above is now described.

**-allwords** Resultfile will include all implication relationships between pairs of terms whose collection frequencies lie within the specified range. This operation is likely to be very expensive.

**-list** Wordlistfile contains a list of words specified by the user. Implication relationships between these terms and any of the terms from the collection (which match the frequency criteria) are sought.

**-saved** Matchsetfile is an option disk file containing a list of matchsets saved previously (possibly in an earlier run) using the **savematchset** command. This can be useful in finding terms which are associated with document features not expressible as simple terms. For example, phrases, proximity relations, and regular expressions. A simple-minded check is made to ensure that the matchset corresponds with the specified superdictionary.

**-phrases** This takes a result file from a previous **implications** command and tries to find useful phrases using pairs (or maybe triples etc of the words found to be associated with a particular term or construct. For example, the matchpoint set arising from a search for the phrase **nuclear power** could be fed into an **implications -saved** run which might

produce a list of words like `uranium`, `fuel`, `rods`, `fast` and `breeder`. All possible ordered pairs of these terms are then checked to see how often they occur adjacently in the text base. If the frequency constitutes a high proportion of their joint occurrence, then this suggests that they may be a genuine phrase. Hopefully this process would throw up phrases such as `fuel rods`, `fast breeder` and possibly `uranium fuel rods`.

The probability threshold is by default 0.2 but may be changed using the `probthresh` variable (specifying a number of thousandths).

## 6 FACILITIES FOR TEXT BASE ADMINISTRATORS

### 6.1 Enabling More Efficient Loading

**Padre** incorporates the ability to compress the data held by each cell after loading and dump it into a file called `cell10`, `cell11`, etc. in a directory specified by the user.

The syntax of the `compress` command is :

```
>> compress "name of non-existent directory"
```

It is the user's responsibility to ensure that sufficient disk space is available for the directory of compressed data. A compression ratio of roughly 2:1 is achieved using the compression algorithm.

In the specified directory will be placed a series of files called `cell10`, `cell11`, ... etc containing the compressed data for each cell and a `.info` file to record the start-of-entry marker.

Data compressed in this way may be loaded into an AP1000 with a different number of cells, but the `loadbalance` command may be needed to ensure efficient processing and use of memory.

**Padre** also allows a loaded text base to be dumped onto cell option disks, to files in the Local or HiDIOS filesystems.

```
>> dodump filestem      - dump to option disk as filestem.tb
```

### 6.2 Removing entries from a text base

*We should probably make the `drop` and `longerthan` commands work on docsets and then things would be more orthogonal.*

The `drop relevant` command removes every document from the collection which has a relevance score (the sum of the positive and negative cumulative relevance measures). On the other hand `drop irrelevant` removes all those with zero or negative relevance scores.

A search command `longerthan` has been defined to allow documents longer than a threshold number of characters to be identified and possibly purged. In the following sequence, documents containing `computer` and those longer than a million characters will have non-zero relevance estimates and will be removed by the `drop relevant` command.

```
>> "computer "  
>> longerthan 1000000  
>> drop relevant
```

### 6.3 Removing components from a collection

The `purge` command takes a start marker and an end marker as arguments. All text in the entire collection between pairs of markers (including the markers) is deleted.

## 6.4 Combining Text Bases

The `merge` command combines the top two collections on the collection stack.

```
>> doload news1      - load textbase news1.tb from the option filesystem
>> load news2        - load textbase news2.tb from the host filesystem
>> cload news3dir    - load compressed textbase news3
>> merge             - combine last two tbs loaded (news2 and news3).
>> dodump news4      - save the result as the option disk file news4.tb
```

## 6.5 Dumping A Text Base

The current collection may be dumped to a host file using the command `dump filestem`.

# 7 DBMerging Simulation

For the purposes of participation in the TREC5 Database Merging track, a couple of facilities have been added to PADRE. These may potentially developed to become applicable in other circumstances as well.

## 7.1 Lightweight Probes

The `probe` command sends a list of terms to all cells. In response, the cells compute local frequencies for each of the terms, for the near relation between the terms, for a conjunctive relation between the terms, and for the total number of documents for which the cell is responsible. All frequencies are in terms of number of documents satisfying the condition.

```
>> probe term1 term2 ...
```

Currently, the `probe` command produces a line of output for each cell:

```
"topic" PROBE("cell") "tot_docs" "near_f" "and_f" "term1_f" "term2_f" ...
```

Eventually, it is likely that it will produce no output but use the information returned by the cells to set a cell mask pattern which may be transmitted to the cells. (See next section.)

## 7.2 Setting Cell Masks

Each cell maintains a cellmask which at present determines whether the cell participates in query processing or not. The `setcellmasks` command causes a cellmask map maintained by the host to be scattered to the cells.

Cell masks are reset so that all cells participate in query processing whenever a minor reset occurs (most commonly due to `topic` command).

Two forms of the `setcellmasks` command are provided:

```
>> setcellmask - scatter previously loaded cellmask map
>> setcellmask c1 c2 ... - Cells in list are turned on, others off
```

## 8 IMPROVING EFFICIENCY

### 8.1 Load Balance - FTS Method only

**Padre** works most efficiently if the document collection is evenly spread across the AP1000 cells. When a collection is loaded or altered, the current load imbalance is displayed. It is the ratio of maximum chunk size to average chunk size. It should be as close as possible to 1.0. If it is too high, load imbalance may be reduced by applying the `loadbalance` command as many times as necessary.

Do not use the `loadbalance` command if the document collection must remain ordered.

## 9 MEMORY RESIDENT INDEX METHOD

To use this method, both the raw text and an index pre-computed by **parson** must be loaded. Like the text, the index can be loaded from hostfile (`loadindex filestem`) or from option disks (`doloadindex filestem`). For example:

```
>> doload wsj90
>> doloadindex wsj90
>> "money "
```

## 10 SUPER DICTIONARY METHOD

Current work on **padre** is oriented toward dramatically increasing the amount of data which can be handled on a given AP1000 configuration, hopefully without excessively reducing performance or flexibility. The approach taken is to extend the number of searching modes which can be performed using an inverted file index and to allow multiple **parson**-built indexes to be accessed in a single search, without needing to load raw text data or the index files into memory on most occasions.

A large number of files are generated for each component textbase, and a multi-union of the information contained in the dictionary files for each textbase is formed, called a super-dictionary (SD). The **parson** program is used to generate the SD and associated files; refer to the **parson** manual for information on how to build them.

The rest of this section assumes that a SD has been built for a collection of textbases. Super-dictionaries can only be stored on the HiDIOS file system at the current time.

### 10.1 Loading a super-dictionary

The **padre** program is started in the usual way. To load a given SD called, say `sd` do the following:

```
>> usesd sd
```

### 10.2 Searching with a super-dictionary

Once loaded, searching with a SD behaves much like searching with normal text, but with the following restrictions.

- Search terms will only match against the start of words; this is equivalent to having `{wsmode start}` set.
- Only words consisting of alphanumeric characters can be found. A single punctuation mark at the end of a word is also acceptable in a word search term, or at the end of the last word in a phrase search term. Although there is currently no mechanism to match punctuation after each word in a phrase, this will be added in the near future.
- No regular expression search terms can be used. A limited form of regular expression matching for single words may be added in future, but is not currently permitted.
- The `bmg2` command is not available. Multi-alternates can be specified with the `anyof` command, although duplicates are not currently eliminated (ie. each alternate in the command must be distinct from all others).

Two variables have particular importance when using SDs. These are the `casesensitive` and `pwprefixmode` variables.

Normally, case sensitive searching is enabled with the `{casesensitive 1}` command. However, if instead the variable is set to 2 then words are matched as if they were proper names - at least the first letter must be uppercase. This mode allows the user to obtain matches for 'BANK', 'Bank', 'BaNk', but fail to match 'bank'. The decision algorithm for determining whether a word is a match is slightly complex, but can be described as follows:

- Search term is all uppercase - only hits of the term with all uppercase characters are matched.
- Search term has first letter uppercase, and rest mixed or lowercase - hits of the term with at least the first letter uppercase are matched.
- Search term is all lowercase - any hit of the term will be matched (as if in case insensitive mode).

This flexibility is especially important when searching for phrase matches.

The second variable of significance is `pwprefixmode`. When set to 0 (the default) phrases must match with at most 2 characters between any word in the phrase. (This mechanism allows the searches to match over inter-word boundaries that are expanded because of a linebreak, but still requires all the words to match exactly.) If instead the variable is set to 1 then words in the phrase search term are treated in a limited degree as prefixes, and will allow matches to occur where the gap between the start of two consecutive words in the phrase is the length of the first prefix word + 5 characters. It is hoped that this will be generalised to perform fully correct prefix matching of phrase word terms at some point in the future.

### 10.3 Restrictions on Use of Super Dictionaries

Only one SD can be loaded at a time; loading one removes any previously loaded collections and indexes.

We believe that it would be possible to switch between different super dictionaries using multiple `usesd` commands.

Once a `usesd` command has been issued, the normal loading commands will load data, but will be ignored in subsequent search commands. This is a deficiency which may be remedied.

A pseudo-collection may however be loaded using the `loaddocs` command after a `used` command. This leaves the super dictionary "loaded" but suspends its use until the next `reset all` command. While in this special mode, new textbases may be loaded and searched using the normal commands.

## A SYNTAX FOR REGULAR EXPRESSIONS

The syntax for the regular expressions in **padre** are the same as the syntax for regular expressions in the public domain GNU egrep program. The following table is taken from the documentation for GNU egrep.

c	a single (non-meta) character matches itself.
.	matches any single character except newline.
?	postfix operator; preceding item is optional.
*	postfix operator; preceding item 0 or more times.
+	postfix operator; preceding item 1 or more times.
	infix operator; matches either argument.
^	matches the empty string at the beginning of a line.
\$	matches the empty string at the end of a line.
\<	matches the empty string at the beginning of a word.
\>	matches the empty string at the end of a word.
[ <i>chars</i> ]	match any character in the given class; if the first character after [ is ^, match any character not in the given class; a range of characters may be specified by <i>first-last</i> ; for example, \W (below) is equivalent to the class [^A-Za-z0-9]
( )	parentheses are used to override operator precedence.
\ <i>digit</i>	\ <i>n</i> matches a repeat of the text matched earlier in the regexp by the subexpression inside the <i>n</i> th opening parenthesis.
\	any special character may be preceded by a backslash to match it literally.

(the following are for compatibility with GNU Emacs)

\b	matches the empty string at the edge of a word.
\B	matches the empty string if not at the edge of a word.
\w	matches word-constituent characters (letters & digits).
\W	matches characters that are not word-constituent.

Operator precedence is (highest to lowest) `?`, `*`, and `+`, concatenation, and finally `|`. All other constructs are syntactically identical to normal characters.

(Note : This appendix is taken from the gnu grep manual, and the source for the regular expression search routine is taken from the gnu “Extended Regular Expression Matching and Search Library”, both of which are Copyright © Free Software Foundation, Inc.)

## B SUMMARY OF COMMANDS

### Loading Textbases

load <i>filestem</i>	load <i>file.tb</i> into memory.
cload <i>directory</i>	load compressed files from <i>directory</i> into memory.
cload <i>directory n</i>	load every <i>n</i> th compressed files from <i>directory</i> .
cload <i>file</i>	load compressed files listed in <i>file</i> into memory.
doload <i>file</i>	load data from <i>file</i> on option disk filesystem.

### Dumping Textbases

dodump <i>file</i>	dump current collection to <i>file</i> on local filesystem.
dump <i>file</i>	dump current collection to <i>file</i> on host filesystem.
compress <i>directory</i>	compress current <b>padre</b> textbase into cell files in <i>directory</i> .

### Manipulating Textbases

merge	replace two top collections with a single merged collection
longerthan <i>n</i>	Find all documents at least <i>n</i> characters long
drop relevant	remove documents with positive relevance.
drop irrelevant	remove documents with negative or zero relevance.
purge <sm> . . <em>	remove text between markers.
loadbalance	Balance distribution of data across cells.

### Pattern Matching

<i>pattern</i>	Search for words starting with <i>pattern</i> .
" <i>pattern</i> "	
regexp <i>pattern</i>	Search for a regular expression specified by <i>pattern</i> .
anyof <i>pattern1 pattern...</i>	Search for any one of the <i>patterns</i> .
bmg2 <i>pattern1 pattern...</i>	Search for any one of the <i>patterns</i> .

### Proximity Relationships

fby <i>n</i>	Replace top <i>n</i> sets with result set of prox operation.
not fby	ditto
not pby	ditto
near <i>n</i>	ditto
znear <i>n</i>	ditto (but special relevance calculation performed.)
not near	ditto

### Set Operators

union <i>n</i>	Replace top <i>n</i> sets with their union.
intersect <i>n</i>	Replace top <i>n</i> sets with their intersection.
diff <i>n</i>	Replace top <i>n</i> sets with their set difference.
~	Replace top set with set of docs not containing matches.
infix operators: +, ^, -, ~	union, intersect, diff, negate.
use matchsetfile <i>filestem</i>	Select option disk file for savematchsets
savematchset <i>name</i>	Save current set in option disk file as name.
loadmatchset <i>name</i>	Load match set name from option disk file.
matchpreceding <i>regex</i>	Replace each match with preceding instance of regex. Elim. dup.s
matchpreceding sent para doc	Ditto but use predefined patterns for sentence, para or document.

## Component Operators

name = component <sm> . . <em>	Define a component of text that starts with <sm> and ends with <em>.
<i>p</i> within component <i>c</i>	Search for pattern <i>p</i> within specified components.
component <i>c</i> including <i>p</i>	Search for components of type <i>c</i> that include pattern <i>p</i> .

## Lexicographic Output

pr	Print the last set of matches.
sample	Print a sample of the last set of matches.
save	Save the last set of matches to a file.
shift. <i>n</i>	Shift the last set of match points to the right <i>n</i> characters.
shift.- <i>n</i>	Shift the last set of match points to the left <i>n</i> characters.
context <i>m n file</i>	Put <i>m + n</i> words of context for each match in <i>file</i> .

## Relevance Ranking and Output

exclude <i>biased current others  - t  + t</i>	Mandatorily exclude documents from ranking.
include <i>current others  - t  + t</i>	Mandatorily include documents in ranking.
topic <i>topicname</i>	Reset cumulative relevance metrics and flags.
reset all	Remove pseudo-collection, matchsets and reset all relevance records.
reset bits	Reset relevance bits only.
reset start	Make everything the way it was when padre started.
reset vol	Clear all volatile sets from the match set stack.
reset weights	Reset cumulative relevance metrics only.
top <i>n</i>	list <i>n</i> Most relevant documents.
getdoc <i>doc-id</i>	Display specified document.
retrieve <i>n</i>	Retrieve <i>n</i> most relevant documents into a file.

## Distance Based Relevance

span all <i>n w l m</i>	relevance due to <i>m</i> or more of top $n \geq 1$ sets, weight <i>w</i> , prox. limit <i>l</i>
span leading <i>n w l m</i>	as for all but partial spans of length <i>p</i> must include first <i>p</i>
span key <i>n w l m</i>	as for all but partial spans must include first <i>k</i>
znear <i>n</i>	restricted form of span. Also computes a result set.

## Operations on doc sets.

matchdocstarts	Convert top matchset into a docset.
makedocset current	Equivalent to matchdocstarts.
makedocset relevant	Make a docset of all relevant docs.
makedocset <i>mn</i>	Make a docset of <i>n</i> most relevant docs, excluding first <i>m</i> .
makedocset <i>n</i>	Make a docset of <i>n</i> most relevant docs.
makedocset included	Make a docset of all included docs.
makedocset excluded	Make a docset of all excluded docs.
finddocs <i>docid...</i>	Make a docset of the docs whose ids are listed.
findfiledocs <i>file</i>	Make a docset of the docs whose ids are listed in the specified host file.
listdocs <i>file</i>	Put docids of all docs in top set in <i>file</i> .
withindocs	matches in second top set are removed if they are not in docs specified in t
wordlist <i>file</i>	Put list of all unique wds (with doc count) in docset in <i>file</i> .

wordlist2 *file* As for wordlist but for top two docsets.  
loaddocs Load all documents referenced in top docset as a pseudo-collection.

### Selecting MRI or SD Methods.

loadindex *filestem* Load pre-computed index from host and use MRI.  
doloadindex *filestem* Load pre-computed index from option disk and use MRI.  
usesd *filestem* Load pre-computed super dictionary from option disk and use SD.

### Term Co-occurrence

implications -allwds *SD lo hi rsltfl* - all words with suitable freq.s.  
implications -list *SD wdlst lo hi rsltfl* - specified words and other words.  
implications -saved *SD matchfile lo hi rsltfl* - specified matchpoints and other words.  
implications -phrases *SD oldrsltfl rsltfl* - find phrases in lists of words in previous rsltfl

### DB Merging

probe *t1...* Send list of terms to all cells and display cell-by-cell doc freq.s  
setcellmasks Scatter host-held cell map to cells to turn cells on/off.  
setcellmasks *c1...* Turn specified cells on, others off.

### Customise Operation of padre.

{*variable new-value*} Change value of *variable* to *new-value*.

### Exit

quit Quit padre.

Not all of the above are yet implemented, see main text for details.

## C SUMMARY OF VARIABLES

Below is a list of variables whose values control **padre**'s behaviour. In most cases default values are listed.

- `printcontextlength` <number> (default 74)
- `printbefore` <number> (default 14)
- `samplesize` <number> (default 10)
- `savefile` <filename>
- `proximity` <number> (default 200)
- `base` <number>
- `setname` <name of set, starting with a letter>
- `casesensitive` 0 | 1 | 2 (default 0)
- `weight` <number> (default 5)
- `mode` `par1` | `trec` | `dflt` | `test` | `dbm` (default `dflt`)
- `wsmode` `start` | `any` (default `start`)
- `pwprefixmode` 0 | 1 (default 0)
- `relmode` `wei` | `boo` | `noc` | `squ` | `pro` | `zmo` | `cou` (default `weighted`)
- `aomode` `onepass` | `sdcompat` (default `onepass`)
- `zmode` <six-digit number> (default 01 - ANU TREC4) (33 - UW TREC4)
- `zmode` 1st digit - word count method 0 - `dumb`; 1 - `sophist`.
- `zmode` 2nd digit - partial span score 0 - `lower than any longer`; 1 - `0.1 * longer`
- `zmode` 3rd digit - decay function 0 - `num/denom`; 1 - `normal`; 2 - `exp decay`; 3 - `custom`
- `zmode` 4th digit - adaptive 0 - `no`; 1 - `yes`
- `zmode` 5th digit - numerator 0 - `1.0`; 1 - `hits/spanlen`; 2 - `hits`
- `zmode` 6th digit - denominator 0 - `len`; 1 - `sqrtlen`; 2 - `maxlen`, 5
- `probthresh` <number> (default 500) - representing thousandths

## References

- [1] Fawcett, H., *PAT 3.3 User's Guide*, University of Waterloo Centre for the New Oxford English Dictionary, Waterloo, Ontario, Feb 1991
- [2] PADRE WEB page, [http://cap.anu.edu.au/cap/projects/text\\_retrieval/](http://cap.anu.edu.au/cap/projects/text_retrieval/)