Sino - A Text Search Engine

Andrew Mowbray

Australasian Legal Information Institute
Faculty of Law, University of Technology, Sydney

Version 3.1.17 January 4, 2008

Contents

Chapter 1

Introduction

Sino (short for "size is no object") is a high performance free text search engine. It was originally written in 1995 and has been mainly used to provide production level search facilities for most of the Legal Information Institutes that form part of the *Free Access to Law Movement* (see <http://www.austlii.edu.au> and <http://www.worldlii.org>).

The code in this release is a major rewrite that makes sino much faster and adds new functionality. Sino's features include:

Speed

Sino is **very** fast both in retrieval and indexing times (see Performance below)

Flexibilty

Sino is easy to interface with via a simple C/Perl API as well as a ready written interactive interface for testing or for actual use on sockets etc.

Portability

Sino is relatively small and easy to understand (about 12K lines of C code). It is ANSI/POSIX.1 compliant and can be compiled 32 or 64-bit. Sino concordances (indexes) are portable across platforms with different architectures.

Reliability

Sino has been in use on a number of major web sites answering many millions of requests for the past 10 years and so is robust and reliable.

Free

Sino is open source and is licensed under the GNU General Public Licence (GPL).

1.1 System Requirements

Sino has been built using Solaris (Versions 9 and 10 / Sparc and x86). It has also been tested for FreeBSD (gcc 32 and 64-bit), Linux (gcc 32-bit only) and Windows XP (32-bit only using cygwin and VC++). It is written in a portable way and ought to run on most other platforms with little or no change.

In order to perform well, the indexing side of Sino needs a machine with at least 1G of memory. A machine with 4G will deal with most reasonable physical concordances (up to about 50G of text per physical concordance).

1.2 System Overview

Sino consists of two parts: the indexing software and the search engine itself. The indexing tool is called **sinomake**. At its simplest, **sinomake** reads a directory hierarchy and creates a Sino concordance (via the command line call that is literally just "sinomake"). Like most text search engines, the concordance consists of an *inverted file* that matches words to word occurrences and is stored as a set of files (called **.sino_words** (the dictionary or *lexicon*), **.sino_hits** (the actual "word occurence" information) and up to about a half dozen or so other files that deal with more mundane but important matters such as *database masks*, *named sections* and *synonyms*). These typically total about 40% of the size of the indexed files with the default common word list or up to 65% with no common words (if you were wondering where the name "sino" came from, it is meant to essentially convey that it is a tradeoff between disk space and speed. See <http://www.austlii.edu.au/austlii/help/sino.htm> for an historical background).

sinomake can also be used to incrementally update physical concordances (see the **-a**, **-u** and **-z** flags below).

One of the most powerful Sino features is support for *virtual concordances*. These allow you to list the names of physical concordances in a file (called **.sino_database**) and have them function as though they were a real concordance. This makes managing big collections easier. (It is particularly good at AustLII as we use this to share concordances between different systems eg AustLII has all of the Australian and New Zealand materials, NZLII just has the New Zealand content, CommonLII incorporates both and WorldLII has this plus pretty much every other country in the world!). There is no significant search speed penalty in doing this.

Most of the rest of the code constitutes the search engine itself. This is best accessed via the Sino API from C/C++ or Perl. Only three call are necessary for a pretty good system. Make it 5 and you are close to full functionality (with database masking and so forth).

There is also a free standing program (called **sino**) that can interactively search the created index (or virtual indexes) and if needs be can be invoked directly from a CGI (please don't do this unless you really have to - the overhead in starting up a new process is sort of scary if you analyse it properly).

1.3 Getting Started

To start using Sino, compile the distribution. Sino has been most thoroughly tested under Solaris and (and to a lesser extent) Linux, FreeBSD and Windows NT, but the code is quite portable and should work in most modern environments.

1.3.1 Installation

Firstly, unpack the distribution:

```
gunzip sino-3.1.11.tar.gz
tar xvof sino-3.1.11.tar
```

Any version of tar should work and there is no need to install specialised configuration software. If you are installing Sino in a Unix-like environment you can use the autoconf generated "configure" program to modify the Makefile in the home Sino directory as in:

```
./configure
```

If you intend using the Sino API with Perl under Unix, the simplest way to make sure that your libraries are compatible is to ensure that the right target version of "perl" is the first one of the current "PATH" and typing:

```
make perl
```

This will compile Sino, the Sino API and supporting libraries for Perl with the same compiler, flags and libraries that were originally used to build the "perl" binary.

If you are not using Perl (or just want greater control over what is happening) you can use one of the following:

```
make          # generic C Compiler / POSIX OS
make perl     # get options from perl / build perl libraries
make gcc      # 32-bit GCC C Compiler / Any OS (eg BSD/Linux)
make gcc64    # 64-bit GCC C Compiler / Any 64-bit OS (ditto)
make sungcc   # 32-bit GCC C Compiler / Solaris OS
make sungcc64 # 64-bit GCC C Compiler / Solaris 10+ OS
make suncc    # 32-bit Sun Sudio C Compiler / Solaris OS
make suncc64  # 64-bit Sun Studio C Compiler / Solaris 10+ OS
make hpcc     # 32-bit HP-UX C89 Compiler / HP-UX
make hpcc64   # 64 bit HP-UX C89 Compiler / HP-UX
nmake msc     # Microsoft C (VC++) / Microsoft Windows
```

If you are in a non-POSIX OS environment and nothing else works, try:

```
make ansi
```

If this still doesn't work, it means that your compiler will not even compile code following the original ANSI standard (now well over a decade old)! At this stage, if you are confident that you know what you are doing and still can't solve the problem send me mail (andrew@austlii.edu.au).

The build process puts the output binaries and libraries in the directories "bin" and "lib". You can then manually copy these to somewhere in everyones' path or if running as root on a Unix box and want to copy them to "/usr/local/bin" and "/usr/local/lib" type:

```
make install
```

The binaries do not need anything to be in any particular place. If you are using Perl the "sinoAPI.so" and "sinoAPI.pm" files should be in the same directory as ".pl" programs that use them or in the $(INC) path as reported by "perl -V".

1.3.2 Testing

Once you have Sino installed, you can make a Sino concordance (index) by setting the target directory hierarchy as the default directory to be indexed and invoking **sinomake**. For testing purposes, try to make this something that is not too large or it will take a while.

It is probably a good idea to put the **-V** flag on to see what it is doing. Type:

```
cd the/directory/to/be/indexed
sinomake -V
```

If you don't have write permissions to this, use a symbolic link.

Assuming all goes well, you can start searching the index with the **sino** utility. Type:

```
sino
```

It should respond with a "sino>" prompt. Type the command:

```
search banana
```

Obviously it is a good idea to replace the word banana with something that is likely to be in the data that you have indexed (but "banana" has become a bit of a word of choice that tends to crop up in most places :-). You should get back the number of search results and then pairs of lines listing the file name and title (if any - else the name of the file again) of the retrieved documents.

If all this works, Sino has been successfully installed!

Chapter 2

Building and Managing Concordances

This section details how concordances are built and configured.

2.1 Invoking sinomake

As we saw in the previous chapter, files are indexed via the utility - **sinomake**. By default, **sinomake** will index all files in and below the current directory. You can specify a different starting directory (and optionally a target directory) as in:

```
sinomake home/andrew/src-dir /home/raid2/live/target-dir
```

You can also add a third argument to specify where the configuration files are as in:

```
sinomake src-dir /home/raid2/live/target-dir config-dir
```

2.2 Incremental Updating

Once built, Sino concordances can be updated by calling **sinomake** with either the **-a**, **-u** or **-z** flags. The simplest of these is **-u** which looks for files that have been added, deleted or modified and updates an existing concordance. It is invoked as:

```
sinomake -u
```

This is generally much faster than doing a rebuild. The **-a** option appends to an existing concordance. Files to be appended are listed in the .sino_files file. This is faster again (as it doesn't involve sifting through a whole directory hierarchy to see what has changed).

When updating, **sinomake** usually checks file sizes and modification times. You can skip the modification time check and just rely upon file sizes by using the "-z" flag as well or instead of "-u". At AustLII, this is helpful as we often get weekly dumps of an entire database.

2.3 Document Order

Documents are processed and stored in alphanumerical order (useful to save having to sort search results). Unlike normal alphanumerical sorting, numerical substrings sort in numerical order (that is, "s1.html", "s2.html" and "s10.html" will sort as "s1.html", "s2.html" then "s10.html"). You can invert this by putting an empty file named **.sino_reverse** in any source directory or sub-directory. In our legal data, we use the default search order for legislation (which is alphabetic) and inverse order for judgments (which are numerically numbered) to return these in reverse date order).

2.4 Including and Excluding Indexed Files

There will be some files that you probably don't want to index (tables of contents etc). You can control which files get indexed via the files: **.sino_include** and **.sino_exclude**.

sinomake first reads **.sino_include** to see if a file is valid and then **.sino_exclude** to see if it ought to be excluded. Both files should consist of zero or more lines each containing *globbed* expressions. A "globbed" expression is as for file matching in sh(1) (see also Pattern Matching below). This form of pattern matching is widely used within Sino (partly because it is more natural for filename matching and even word matching, but also because pattern matching often is needed in time critical parts of the code and the standard regex is very slow.

Typical **.sino_include** and **.sino_exclude** files for indexing a web site are:

```
# Example .sino_include

*html
*htm
*sino_text
```

```
# Example .sino_exclude

*index*
*OLD*
*BUILD*
```

If you want even greater control, you can tell **sinomake** exactly which files you want to index by creating your own **.sino_files** file and invoking **sinomake** with the **-f** flag as in:

```
find . -name *html -print > .sino_files
sinomake -f
```

Care needs to be exercised not to later invoke **sinomake** without the **-f** flag as it will clobber the **.sino_files** file with its own version.

2.5 Word Handling

The fundamental concordance element in Sino is a *word*. A word consists of a continuous sequence of alphanumeric characters. Non-alphanumeric symbols and anything appearing within angle brackets (ie <tag>s) are not indexed. Words are case insensitive. Alphanumeric characters include all regular [A-Za-z0-9] ASCII symbols and accented letters in the ISO 8859-1 (Western European) character set. Sino can be recompiled for other ISOs. The only one supported in the current code is Greek (ISO 8859-7). For ISO 8859-1, accented characters (and their equivalent HTML entities) are internally converted and stored in their non-accented (ASCII) form. A similar conversion is done on search strings so that a search will work regardless of whether or not an accent is present. For the most part this is transparent to the user and in practice produces more good than bad.

Words may also have derivatives. A derivative is a version of a word that is treated in all respects as being equivalent (including being stored as such). By default, simple English plurals are mapped to their base form. The default mappings are to convert words ending with "ies" to "y" and "es" and "s" to null. This can be disabled or configured to deal with other languages.

At search time, words may also be expanded to include synonyms. These are defined as comma separated lists in the **.sino_synonym** file which is processed by **sinomake** to build the **.sino_synonym_index** file. By default, there are no synonyms.

The result of the above is that the following words will be stored and processed as follows:

cat	cat
123	123
cat123	cat123
CAT	cat
cats	cat
cat's	cat s
cãt	cat
c&aacute;t	cat

2.5.1 Common Words

Some words may be designated as being common (sometime called "stop-words"). A *common word* is a word that is not indexed or searchable. Typically, this is used to remove very commonly occurring and non-informative words from searches and the index to improve search times, indexing times and to reduce index sizes. The savings are not huge and you shouldn't feel that you have to go overboard with these (and in fact, removing them altogether is not such a bad idea).

In searches, a common word serves as a place holder for any word. Unless disabled or configured, the default list of common words is:

```
be because been before being but by can could did do
does ever following for from given had hardly has have
having he hence her here hereby herein hereof hereon
hereto herewith him his however if in into is it its
may me member more must my no nor not of on onto or
other our out shall she should so some such than that
the their them then there thereby therefore therefrom
therein thereof thereon thereto therewith these they
this those thus to too under unto up upon us very viz
was we were what when where whereby herein whether which
who whom whose why will with within would you your
```

Most of the above is configurable. A new set of common words can be specified via the **.sino_common** file. Without the use of any **.sino_common** files, **sinomake** will index all words and **sino** or the API will use the default set unless a search word is in quotes.

The **.sino_common** file consists of zero or more words separated by white space. The **.sino_common** file is used by both **sinomake** and **sino** or the API.

2.5.2 Derivatives

The derivative behaviour can be altered via the **.sino_derivatives** file. This file consists of zero or more lines each with either a suffix and replacement or a globbed pattern and replacement text. To specify a set of derivatives for English and Portuguese for example, you might use something like:

```
#  Default English derivatives

ies y
es
s

# Portuguese derivatives

mães mãe
ãos ão
ães ão
ões ão
ços ço
...
```

It is much faster to use simple suffix replacement text pairs as in the example. If you really need to use pattern matching see the **sino**_derivatives(5) manual page.

2.5.3 Synonyms

Synonyms are defined via the **.sino_synonyms** file. It consists of zero or more lines each with a comma separated list of words and/or phrases. For example:

```
unsw, university of new south wales
small, tiny, little
...
```

sinomake normally indexes this file as part of its normal operation and produces a file called **.sino_synonym**_index. You can force **sinomake** to just remake the synonym index with the -S flag.

2.6 Document Information

Sino is designed to work with unstructured documents that require no modification. Nevertheless, if you are building a new database you may wish to specify additional information within documents. Sino allows you to do a number of things in this regard including provision of a document title and date as well as dividing documents into named sections.

2.6.1 Document Tags

Sino gathers the title of a document from the contents of the first <title> ... <title> pair. This is obviously designed for html and you will need to place these tags within a comment for other applications. Document dates are specified with the <!–sino date date–> tag (where date is a date in any sensible English dd/mm/yy format). Named sections are introduced via the <!–sino section section–> tag (where section is the name of the named section). You can index hidden text with the <!–sino hidden text–> tag.

Putting these together, a fully worked up file might look like:

```
...
<title>R v. Smith</title>
...
<!--sino hidden docket-no-1234-->
...
<!--sino date 1 March 2006-->
...
<!--sino section catchwords-->
Murder . Defence of provocation
...
<!--sino_section judgment-->
This defendant is charged under .
...
```

2.6.2 Shadow Files

One final document related facility is provision for shadow files. These are used when you want to index non-ascii files (such as pdfs). Where a file has a **.sino_text** suffix, at search time only the file name prefix will be returned.

To make this work, you need to convert all of the target files to text and include these in the files to be indexed. For example:

```
find . -name "*pdf" -exec pdftohtml {} {}.sino_text \;
sinomake
```

and include the following line in **.sino_include**:

```
*.sino_text
```

2.7 Virtual Concordances

When managing large collections, you may wish or need to use the virtual concordance feature. This allows for faster and more convenient updates and provides for concordances that would otherwise be larger than maximum file size (for 32-bit versions).

You do this by listing a number of physical concordances in a file called **.sino_database**. This file consists of one or more lines each listing the directory for a physical concordance. These do not nest (ie there should only be one **.sino_database** file). In order for database masking to work properly (see below) all physical concordances should be relative to (ie below) the directory in which **.sino_database** exists. As a special case, "." refers to a physical concordance in the same directory as **.sino_database**. This is useful if you are maintaining an "updates" concordance which may contain documents from anywhere in the directory hierarchy.

For example:

```
# Example .sino_databases file


au/cases
au/legis
nz
```

The only other files that need to be in directory containing **.sino_databases** are: **.sino_derivatives**, **.sino_synonym**_index, **.sino_common** and **.sino_sections** (if any of these are in use). You should make sure that all component databases have been built using the same **.sino_common**, **.sino_derivatives** and **.sino_sections** and that the latter two match the ones in the **.sino_database** directory.

Virtual concordances are almost as fast as real ones, and it is OK for the number of separate physical concordances to be large (at least into the hundreds).

2.8 Database Masks

Masks allow you to restrict searching to parts of a database. These are very efficiently implemented and their use is encouraged over having totally separate real databases.

2.8.1 Directory Masks

A mask is a directory prefix relative to the database location. For virtual databases, this should be relative to the directory containing **.sino_database**.

On AustLII, we use this to create the illusion of separate collections for different courts and legislatures (using masks like "au/cases/cth/HCA" and "au/legis/cth/consol_act").

Masks can be set in several ways. The most usual way of doing this is via the **sinoAPI_set_masks**(3) call. Using the API is not covered until the next chapter, but here is an example of how to do this anyway:

```
#include "sinoapi.h"

#define DATABASE          "home/www"

int    nmask = 3;
char    * masks[3] = {     "au/cases/cth/HCA",
"au/legis/cth",
"au/other/HCTRANS" };

main()
{
if (sinoAPI_set_database(DATABASE) == -1) {
       printf("can't open database\n"); exit(1);
}
if (sinoAPI_set_masks(DATABASE, nmask, masks) == -1) {
    printf("can't set masks\n"); exit(1);

/* go on and search / display results ... */
}
```

2.8.2 File Masks

Sino also supports "file masks". *File masks* allow you to specify a set of globbed patterns which will be used to restrict documents returned. These are much less efficient than proper mask paths and should only be used when absolutely necessary. They are set with the **sinoAPI_set_fmasks**(5) call which has the same syntax as **sinoAPI_set_masks**(5).

Chapter 3

Interfacing with Sino

Once you have created a Sino concordance, the next step is to create an interface to the search engine. There are a number of ways of doing this but the preferred method is to use the API. (The other methods are to put the **sino** binary on a Unix socket or to call this directly from a CGI program, which is not encouraged).

There are three basic calls that your interface needs to make to the API: one to *set the database*, one to *do the search*; and then a loop of calls to *get the results.* The routines are called: **sinoAPI_set_database**, **sinoAPI_search** and **sinoAPI_get_next**. Sino has a complete set of manual pages that give all the gory details, but a simple example at this point is probably the most useful.

3.1 Interacing from C / C++

For C or C++:

```
#include <stdio.h>
#include <sinoapi.h<


#define RANK        1
#define SYNONYMS    0
#define DATABASE    "home/www"


main()
{
    char            search[MAXTERM],
                    file[MAXTERM],
                    title[MAXTERM];
```

```
        unsigned long   score,
                        date,
                        size,
                        ndocs;


        if (sinoAPI_set_database(DATABASE) == -1) {
            printf("can't open database\n"); exit(1);
        }
        for (;;) {
            printf("search: "); gets(search);
            if (sinoAPI_search(search, RANK,
                      SYNONYMS, &ndocs) == -1) {
                printf("search failed\n.); continue;
            }
            while (ndocs-- > 0) {
                if (sinoAPI_get_next(file, title, &score,
                                       &date, &size) == -1)
                    break; /* failed */
                printf("%s [%ld]\n", title, score);
            }
        }
    }
```

This example shows a number of things: We've selected the database directory (containing the **.sino_xxx** files we generated with **sinomake**) via the call: **sinoAPI_set_database**(DATABASE). Then we have done the actual search with a call to **sinoAPI_search**(search, RANK, SYNONYMS, &ndocs).

search is a string containing our search (see below for search synyax), *RANK* and *SYNONYMS* are Boolean values saying whether or not we want ranked results and synonyms employed and *ndocs* is a return value to tell us how many documents were found.

Finally, we call **sinoAPI_get_next** *ndocs* times or until it fails (which it shouldn't unless we call it too many times!). It gives us five bits of information about the retrieved document: its file name (*filename*), the title of the document (*title*), the rank (*score*), the date (*date*) and the file size (*size*). Hopefully most of these are pretty obvious, but again in the interests of keeping things simple we will leave a detailed discussion of what these mean and how they are gathered until later.

3.2 Interfacing from Perl

You can do the same sort of thing in perl with a script like this:

```
#!usr/bin/perl

use sinoAPI;

$RANK = 1;
$SYNONYMS = 0;
$DATABASE = "home/www";

sinoAPI::sinoAPI_set_database($DATABASE) != -1
                     || die "Can't open database";
$ndocs = $score = $date = $size = 0;
while (TRUE) {
    print "Search: ";
    ($search = <STDIN>) ne "" || exit;
    ($status, $search, $ndocs) =
        (sinoAPI::sinoAPI_search($search, $RANK, $SYNONYMS);
    if ($status == -1) {
        print "bad search\n"; next;
    }
    print "$ndocs documents found\n";
    while ($ndocs-- > 0) {
        ($status, $filename, $title, $score, $date, $size) =
            sinoAPI::sinoAPI_get_next();
        if ($status == -1) {
            die "sinoAPI_get_next failed\n";
        }
        print "$filename: $title (Score=$scoreSize=$size)\n";
    }
}
```

3.3 Other Programming Languages

If you are programming in something other than C or Perl, most languages have some way of interfacing to C. (**Swig** (*Simplified Wrapper and Interface Generator* is very helpful in this regard. If you write something, please send it to me and I'll put it in the offical release.

3.4 Calling sino Directly

If you want to call the **sino** application directly see the sino(1) man page. **sino** has a well structured command language and set of return messages that make this pretty easy (and it's the way we used to do it many years ago).

The above examples are in the distribution in the "examples" directory.

Chapter 4

Searches

Sino has a flexible search parser that attempts to deal with connectors from systems like Google, Lexis and WestLaw. The formal Booelan syntax is set out in an Appendix.

4.1 Words

As per the above, the search parser allows for Unix shell style ("globbed") pattern matching of words. The following wild cards are recognised:

```
*        matches any string (including null)


?        matches any single character


[ ... ]  matches any one of the enclosed characters. A
         pair of characters separated by a '- ' matches a
         range of characters (eg [a-c] will match 'a', 'b',
         or 'c'). If the first character is a '^' or a '!',
         characters not enclosed are matched (eg [^a-c]
         will matched anything except 'a', 'b' or 'c'.
```

A pattern must match an entire word. To search for words containing substrings, use "*substring*". The left square bracket symbol is also used for boolean grouping. Where you wish to start a word with a [...], you need to put the whole word in quotes (eg "[ab]*ing"). As far as is consistent, Sino also supports regular expressions. It will for example, treat the sequence ".*" as "*", ignore '^' and '$' characters and will even deal with agrep's '#' character. The main limitation is that sequences such as "[0-9]*" will not work.

4.2 Phrases

A phrase is just a sequence of words. Phrases may be enclosed in double quotes, but this is not usually necessary. It is useful to turn any operators into regular words (eg "goods and services") and to enforce searching of common words that are indexed by **sinomake** but excluded at search time (by including a **.sino_common** file in the same directory as the physical concordances).

4.3 Boolean Operators

Words and phrases may be connected together with boolean and proximity operators to form more complex searches. The operators are borrowed from a number of existing search engines (Google, Lexis, Westlaw, QL etc). They may be used in any combination and regardless of their heritage.

4.3.1 Boolean AND

The boolean AND operator allows you to identify documents which contain two (or more) words or phrases. It may be written as: "and", '+', '&' or "&&". Some typical searches are:

```
copyright and material form
18 and crimes act 1900
defamation and journalist and newspaper
```

Where the keyword "and" is used to indicate a boolean AND it has low precedence (like on Lexis) - it is only evaluated after both of its arguments have been fully evaluated. The rationale for this is that OR is usually used for synonyms which ought to group tightly and so giving AND a lower precedence is usually more convenient for free text searching and is less likely to lead most folk into difficulties.

4.3.2 Boolean OR

The boolean OR operator is used to find documents containing either or both of two terms and is typically used to find synonymous words and phrases. It is written in Sino as: "or", '|' or "||". Examples include:

```
section or s
husband or wife or spouse
proprietary limited or p l or pty ltd
```

4.3.3 Boolean NOT

The NOT operator allows you to find documents which contain one thing but not another. It may be written as: "not", '-' or "%" In practice, this operator is seldom used, but to illustrate:

```
trust not family
trade practice act not 51
```

4.4 Proximity Operators

Proximity operators are used to find documents where two or more terms appear near each other or in particular parts of a document. Sino indexes documents in terms of where words appear (it does not record paragraph or sentence information). Consequently, all proximity operators are in terms of word positions. The simplest form of this class of operators is "near" (or /s or /p). This operator requires that words or phrases appear within 50 words of each other. For example:

```
smith near brown
31 near bail act 1900
```

Although convenient, this operator is obviously a little on the restrictive side. For more flexible proximity searching, you can use the following operators (or their Lexis or Westlaw equivalents):

`/n/`	`within n words`
`/m, n/`	`within n - m words`
`w/n`	`Lexis equivalent to /n/`
`/n`	`Westlaw equivalent  to /n/`
`pre/n`	`Lexis equivalent to /1,n/`

For example:

```
smith w/10 brown>
smith /10/ brown
smith /-10,10/ brown
smith pre/10 brown
smith /1,10/ brown
```

4.5 Named Sections

Named section searching takes the following form:

```
section(searchterms)
```

Standard named sections are **title** (the html title of a document) and **text** (everything). Sino also recognises 1 as a Lexis compatibility equivalent for title.

4.6 Dates

Date searches take the following forms:

```
date = date
date < date
date > date
date >< date1 date2
```

Any sensible (dd/mm/yy ordered) date is OK. Month names may be in English, French, German, Spanish or Portuguese. For example;

```
date = 3/12/06
date < 31-2-2006
date > 31st February 2006
date >< February 31, 2006 1.1.2006
```

Date restrictions can also be set via the **sinoAPI_set_dates**(3) call. For example:

```
sinoAPI_set_dates(20050101, 20051231);
```

4.7 Precedence

Normally searches are evaluated from left to right. This is subject to the following order of operator precedence (highest to lowest):

```
word
(terms) phrase
w/n pre/n /n/ /m, n/ @
or | ||
and not % && &
```

You can use parentheses eg "(father or mother) and murder" to alter this. If you need to make any special symbols literal, these should be enclosed in double quotes.

Chapter 5

System Limits and Performance

5.1 System Limits

The Sino concordance format is fundamentally 32-bit based (with some 64-bit extensions that are recorded in 32-bit format). The format is the same regardless of whether a concordance is built with a 32-bit or 64-bit version of **sinomake** and is always big-endian (with transparent translation for little-endian machines).

The only hard limits for physical concordances are currently 16M words per document and 64 named sections per database. There is no maximum document limit when using virtual concordances, but there is a limit on physical concordances due to 32-bit indexing of the **.sino_docs** file. This depends upon the size of filenames and document title and on the fastest machine with the best i/o would take about 10 hours of concording to reach this (or about 160G / 15M files of text extrapolating from the AustLII file size / title size mix). This will probably be removed in a future release as hardware processor and i/o times improve.

5.2 Performance

Sino performance levels depend a lot on your platform and in particular how fast your system's I/O is and to some extent how many processors you have.

5.2.1 Indexing Performance

The AustLII system runs on a mid-range Sun Server (12 x UltraSparc IV+ processors with 96G memory with gigabit attached EMS NS 700 NAS storage running Solaris 10). The indexed database consists of 1.1M html files with a mean avaerage size of about 10.5K and totalling about 11.5G.

In this environment, we get around 7.4G per hour over NAS (1h 33m) to build a single physical concordance. Using the same data, but on direct attached storage the rates / times are 15.5G per hour (45m). The lexicon and hits buffer peaks at around 1.4G of memory. (We don't actually usually build the concordance this way, but use a virtual database with 5 physical concordances and dispatch concurrent sinomakes to build these.)

More generalised tests have been done on the searchable HTML content from the BAILII (British and Irish Legal Information Institute). This constitutes 332,000 files totalling 3.6G (mean average file size being 10.8K). The output index is 1.5G (41%).

This table lists the elapsed time taken by sinomake to produce a single physical concordance for the entire database. Results are listed worst (slowest) to best:

6. PC 1 x Dual-core 2.4GHz Athalon processor, 2G memory, IDE disk Fedora 5 (32-bit)	29m 55s	7.0G/hour
5. Dell D600 Laptop 1 x 1.6GHz Intel M processor, 1G memory, IDE disk, FreeBSD 6.1	25m 53s	8.1G/hour
4. Sun PC 2 x Single-core 2.7GHz AMD Operon processors, 4G memory, SCSI disk, Solaris 10	22m 38s	9.3G/hour
3. Sun Fire E2900 12 x Dual-core 1.5GHz UltraSPARC IV+ processors, 96G memory Ultra3-SCSI disk Solaris 10	12m 32s	16.6G/hour
2. PC 1 x Dual-core 2.4GHz Athalon processor, 2G memory, IDE disk FreeBSD 6.1 (64-bit/AMD)	10m 41s	19.7G/hour
1. PC 1 x Dual-core 2.4GHHz Athalon processor, 2G memory, IDE disk Solaris 10	10m 35s	19.8G/hour

In the worst case (a Dell Laptop running FreeBSD or Fedora on a fast box :-), sinomake achieves at least 7G per hour. There should be very little dropoff for much larger databases as the size of the lexicon should be fairly stable for more text (ie there are not a lot of new words left to find). The best platform for sinomake, seems to be a fast commodity games PC running FreeBSD or Solaris giving over 19G / hour (with the larger SPARC box with slower CPUs not to far behind).

The main thing that will defeat sinomake is a lack of physical memory when it is building. The program attempts to match sensible memory sizes to the current environment, but for very small systems (1G or less) this may fail. If **sinomake** is fast and then starts to crawl, it is because the system has started to swap. The solution is to use more virtual concordances or to increase the memory. Remember that if you are using 32-bit that it is only possible to access 4G of memory anyway, but you can always run concurrent sinomakes for components of a virtual concordance. In 64-bit mode, it is theoretically possible to create a single physical concordance of up to several hundred gigabytes, but you really should be thinking about how long this takes and using virtual concordances to deal with static data.

5.2.2 Search Performance

At search time, Sino does not need a lot of memory and retrieval times are less dependent upon underlying hardware and i/o systems. Single word searches return in under 0.050 seconds on virtually any tested platform. Typical searches (a phrase or a simple boolean search) take well under 0.500 seconds.

This table shows the CPU time taken to execute a number of searches. The searches use the most frequently occuring words in the database in an attempt to get a meaniful comparison between machines / operating systems. Times are in seconds:

		Machines / OSs (number as above)					
		Slowest					Fastest
Search	Documents Returned	5	6	3	1	4	2
act	221,054	0.000	0.000	0.000	0.000	0.000	0.000
act AND section	146,645	0.460	0.440	0.470	0.275	0.230	0.240
act OR section	225,075	0.480	0.480	0.420	0.270	0.250	0.187
"high court"	26,041	0.187	0.160	0.140	0.100	0.100	0.062
"pervert the course of justice"	379	0.070	0.080	0.050	0.040	0.040	0.023
		-----	-----	-----	-----	-----	-----
		1.197	1.160	1.080	0.685	0.620	0.512

The slowest searches are predicably for a disjunction of two frequently occuring words on the slowest platforms (the laptop and the fast machine running Fedora). The dual Operton Solaris box does very well and the large SPARC box drops considerably (producing the slowest result for for a conjunctive search).

Chapter 6

Conclusion

More detailed information is contained in the manual pages. If you have any problems that the documentation does not address, email me at <andrew@austlii.edu.au> or send feedback to <feedback@austlii.edu.au>.

Chapter 7

Appendices

7.1 Formal Description of the Search Language

The search language used by Sino attempts to be compatible with a number of common syntaxes (particularly Google, Lexis and Westlaw). The formal syntax for the language is as follows:

```
boolean_expr   =  boolean_term { TOKEN_AND boolean_term }
               |  boolean_term { TOKEN_NOT boolean_term } ;


boolean_term   =  proximity_term { TOKEN_OR proximity_term } ;


proximity_term =  phrase { TOKEN_WITHIN phrase }
               |  phrase TOKEN_AT TOKEN_SECTION ;


phrase         =  word { word }
               |  TOKEN_QUOTE word { word } TOKEN_QUOTE ;


word           =  TOKEN_WORD
               |  TOKEN_LBRACKET boolean_expr TOKEN_RBRACKET
               |  TOKEN_SECTION TOKEN_LBRACKET
                                 boolean_expr TOKEN_RBRACKET
               |  TOKEN_DEQUALS
               |  TOKEN_DLESSTHAN
               |  TOKEN_DGREATERTHAN
               |  TOKEN_DBETWEEN ;
```

The lexical analyser works (a little roughly) as follows:

```
TOKEN_AND       =  "AND" | "and" | '&' | "&&" | '+' ;
TOKEN_OR        =  "OR" | "or" | '|' | "||" | '-' ;
TOKEN_NOT       =  "NOT" | "AND NOT" | "not" | "and not" | '%' ;
TOKEN_LBRACKET  =  '(' ;
TOKEN_RBRACKET  =  ')' ;
TOKEN_AT        =  '@' ;
TOKEN_QUOTE     =  '"' ;
TOKEN_NEAR      =  "NEAR" | "near"
                |  '/' NUMBER [ '/' ]
                |  '/' [ '-' ] NUMBER ',' NUMBER '/'
                |  ( "W/" NUMBER ) | ( "w/" NUMBER )
                |  ( "PRE/" NUMBER ) | ( "pre/" NUMBER )
                |  "/P" | "/p"
                |  "/S" | "/s"
                |  "+P" | "+p"
                |  "+S" | "+p"
                |  "W/P" | "w/p"
                |  "W/S" | "w/s" ;
TOKEN_DEQUALS   =  date_key '=' [ '=' ] date ;
TOKEN_DGREATER  =  date_key '>' [ '=' ] date ;
TOKEN_DLESS     =  date_key '<' [ '=' ] date ;
TOKEN_DBETWEEN  =  date_key '><' date date ;
TOKEN_WORD      =  pattern { pattern } ;
TOKEN_SECTION   =  alphabetic { alphabetic } ;
```

```
pattern         =  letter | '*' | '?' | '!' | pattern_group ;
pattern_group   =  '[' [ '^' | '!' ] pattern_range ']' ;
pattern_range   =  letter { letter | ( letter '-' letter ) } ;
date_key        =  [ '#' ] 'date' ;
letter          =  alphabetic | digit ;
alphabetic      =  'a'|'b'|'c'|'d'|'e'|'f'|'g'|'h'|'i'|'j'|'k'|'l'|
                   'm'|'n'|'o'|'p'|'q'|'r'|'s'|'t'|'u'|'v'|'w'|'x'|
                   'y'|'z'|'A'|'B'|'C'|'D'|'E'|'F'|'G'|'H'|'I'|'J'|
                   'K'|'L'|'M'|'N'|'O'|'P'|'Q'|'R'|'S'|'T'|'U'|'V'|
                   'W'|'X'|'Y'|'Z'|'_' ;
digit           = '0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9' ;
```

7.2 Sino Application Programs Interface

7.2.1 NAME

Sino API – Sino Application Programming Interface

7.2.2 DESCRIPTION

Sino provides an API which can be used with C/C++ and Perl. The API consists of the follow routines:

int sinoAPI_set_database(char * *dir*)

Initialise sino / select a new database (must be the first call to the API)

int sinoAPI_search(char * *search*, int *rank*, int *synonyms*, unsigned long * *ndocs*)

Do a search for *search*, optionally with *rank*ing and *synonyms.* Returns modified search as *search* (which should be at least MAXTERM size) and the number of documents found as *ndocs.*

rank should be one of:

```
RANK_NONE               no sorting (any order)
RANK_INVERSE_SCORES     reverse sort on document scores
RANK_INVERSE_DATES      reverse sort on document dates
RANK_DATES              sort on document dates
RANK_FILES              sort on file names
RANK_TITLES             sort on document titles
RANK_INVERSE_TITLES     reverse title alphabetical order
RANK_SCORES             least relevant first
RANK_INVERSE_FILES      reverse sort on docment filenames
```

int sinoAPI_get_next(char * *file*, char * *title*, unsigned long * *score*, unsigned long * *date*, unsigned long * *size*)

Return the next *file* name, document *title*, *score* (ie rank), *date* and *size. file* and *title* should be of size MAXTERM. This routine should only be called *ndocs* times.

int sinoAPI_set_masks(char * *dir*, int *nmasks*, char ** *masks*)

Set up directory masks. Database must be selected and indicated as *dir. nmasks* is the number of masks and *masks* is a vector of directory masks.

int sinoAPI_set_fmasks(int *nmasks*, char ** *masks*)

Set up file masks. *nmasks* is the number of masks and *masks* is a vector of file masks.

int sinoAPI_set_dates(long *d1*, long *d2*)

Set default date restrictions. Both *d1* and *d2* are in ISO format (eg 20060523).

*int sinoAPI_insert_operator(char * from, char * to, char * op)*

Copy the search string contained at *from* to a new one at *to* inserting the operator *op* between each term (word).

int sinoAPI_set_max_results(unsigned long *limit*)

Set the maximum number of results that can be returned to *limit*. Sino will internally rank all results by relevance prior to discarding results past the limit.

unsigned long sinoAPI_search_time()

Return the time it took for the last search (in milli-seconds).

int sinoAPI_sort_results(int *rank*)

Sort the search results in *rank* order. *rank* is the same as for the sinoAPI_search call above.

int sinoAPI_rewind()

Rewind the results position to the first result. The next call to sinoAPI_get_next() will return the first search result. This is equivalent to sinoAPI_set_pos(0).

int sinoAPI_set_pos(unsigned long *pos*)

Set the search result position to *pos*. The next call to sinoAPI_get_next() will return the result at this position. The first result is at position 0.

int sinoAPI_get_database_info(char * *database*, unsigned long * *version*, int * *little_endian*)

Return information about a database. *database* is the directory containing a physical concordance. The routine returns the *version* of **sinomake** that was used to build the concordance and whether or not it is *little_endian.* All modern concordances should be big endian. *version* and/or <little_endian> can be set to NULL if you do want to know their values.

int sinoAPI_get_sino_info(char * *string_version*, unsigned long * *numeric_version*, char * *version_date*, int * *bits64*)

Return information about this version of Sino. *string_version* is string containing the Sino version version number. *numeric_version* is the Sino version number as a numeric (eg 30109). *version_date* is the date of the version. *bits64* is non-zero if you are running a 64-bit version of Sino.

7.2.3 RETURN VALUE

All routines return -1 on failure. The last error in standard format (as per the interactive interface ie: code: level: message) is stored in char * sinoAPI_last_error.

7.2.4 EXAMPLE

```
/*
 *  APIexample.c -- test/demo program for the sino API
 */


#include "sinoapi.h"


#define RANK        1
#define SYNONYMS    0


main()
{
    static  int     report();
    char            search[MAXTERM],
                    file[MAXTERM],
                    title[MAXTERM];
    unsigned long   score,
                    date,
                        size,
                        ndocs;


        /* initialise/select a database - must be call
                                        before other routines */


        if (sinoAPI_set_database("/home/www") == -1) {
                printf("can't open sino database at /home/www\n");
                exit(1);
        }


        for (;;) {
                printf("search: ");
                gets(search);


                /* do a search */


                if (sinoAPI_search(search, RANK, SYNONYMS,
                                                &ndocs) == -1) {
                        printf("search failed (%s)\n",
                                        sinoAPI_last_error);
                        continue;
                }


                /* report results */


                while (ndocs-- > 0) {
                        if (sinoAPI_get_next(file, title, &score,
                                        &date, &size) == -1) {
                                printf("sinoAPI_get_next failed\n");
                                break;
                        }
                        printf("%s [%ld]\n", title, score);
                }
        }
}
```

7.2.5 SEE ALSO

datesapi(3)

7.3 GNU General Public Licence

The software and included in Sino is covered by the GNU General Public linence. The terms of this licence are as follows:

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

7.4 GNU Free Documentation Licence

The documentation included in Sino is covered by the GNU Free Documentation linence. The terms of this licence are as follows:

distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission. B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement. C. State on the Title page the name of the publisher of the Modified Version, as the publisher. D. Preserve all the copyright notices of the Document. E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices. F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below. G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice. H. Include an