THE UNIVERSITY OF QUEENSLAND

Improving the performance of Lucille, a search and indexing library for Python

by
Dan Callaghan

School of Information Technology and Electrical Engineering,
The University of Queensland

Submitted for the degree of Bachelor of Engineering
in the division of Software Engineering
May 2008

18 May 2008

Head
School of Information Technology and Electrical Engineering
The University of Queensland
St Lucia QLD 4072

Dear Professor Bailes,

In accordance with the requirements of the degree of Bachelor of Engineering in the division of Software Engineering, I present the following thesis entitled

'Improving the performance of Lucille,
a search and indexing library for Python.'

This work was performed under the supervision of Dr Peter Robinson.

I declare that the work submitted in this thesis is my own, except as acknowledged in the text and footnotes, and has not been previously submitted for a degree at The University of Queensland or any other institution.

Yours sincerely,

Dan Callaghan

Acknowledgements

I would like to gratefully acknowledge the assistance of my supervisor, Dr Peter Robinson, for his guidance of my research and his input into the preparation of this thesis.

I would also like to acknowledge the help of Sam Kingston.

Abstract

High-level, dynamic, interpreted languages, such as Python, have traditionally been considered viable only for trivial scripting tasks. As the cost of computing power continues to decrease, such languages are becoming practical for a wider variety of programming problems. Nevertheless, for some computation-bound applications the execution overhead of these languages still makes them unfeasible. To mitigate this problem in the case of Python a clear and well-documented C API makes it relatively simple to replace Python code with compiled C code piece-by-piece, at the expense of the expressiveness and flexibility of Python.

The purpose of this project is to investigate these performance issues in the context of an existing code base: Lucille, a Python port of the Apache Lucene search and indexing library for Java.

Initially the performance characteristics of the library were examined, with the aim of identifying possible areas for improvement. A number of techniques were explored for improving the performance of individual components in the library, with varying results. In the report these techniques are generalised, in order to provide guidance to others seeking to improve the performance of Python code.

Contents

List of figures

List of tables

Chapter 1

Introduction

The aim of the project is to investigate techniques for improving the performance of Lucille, a Python port of the Lucene information retrieval library.

Profiling data was collected for the various components of Lucille using Python's builtin profilers [1], in order to identify time-critical sections of the Lucille codebase. This profiling data was then used to inform investigations into improving the performance of these time-critical sections.

The results of these optimisations are presented in this report, along with a thorough description of each technique used and its potential applicability to other Python code.

1.1 Background

1.1.1 Lucene

Lucene (`http://lucene.apace.org/java/`) is an open-source information retrieval library written in Java, developed by the Apache Foundation. It is used by applications which need to perform user searches across large sets of textual data with relevance-ranked results.[1]

Lucene's core functionality is building and searching persistent indexes; it also includes supporting packages for parsing structured user queries, pre-processing documents for indexing (tokenising, stemming, filtering out stop-words), highlighting query terms in results, and other ancillary tasks.

Lucene's search implementation uses the Vector Space Model, introduced for the SMART Information Retrieval System [2]. The model is so called because documents in

[1] A list of applications and Web sites using Lucene is maintained on the Lucene wiki at `http://wiki.apache.org/jakarta-lucene/PoweredBy`.

an index, and user queries against that index, are represented as vectors; the relevance of a search hit is computed as the difference in the angles of the document vector and the query vector.

Various other types of queries and filters are also supported by Lucene, allowing applications to perform complex searches against indexes.

1.1.2 Python

Python is an open-source high-level language and runtime environment, originally written by Guido van Rossum [3], now actively developed by a large community. It is well established as a scripting language and for scientific computation; in recent years it has also gained popularity (along with other dynamic, interpreted languages like Ruby) in the sphere of Web application development.

Python is a substantially different language to Java. Python is dynamically typed and interpreted at runtime, which lies in almost direct contrast to Java; although Java is in some sense interpreted, its interpreter is of a very different style to Python's. The compilation phase in Java is also crucial, since it performs not only static type checking but many other tasks of an ordinary compiler for native code, such as optimisation; Python on the other hand has no compilation phase. The design philosophies underlying the two languages are also vastly different: where Python encourages[2] compact but clear expression, Java favours explicitness and verbosity (ostensibly as a way of ensuring program correctness and minimising programmer error). An excellent comparison of Python and Java is given in [4].

The relative merits of dynamic and static type systems are a topic of perpetual debate, but like so many other "religious wars" in the field, the choice of one over the other is perhaps no more than a question of personal style or preference. Nevertheless dynamic, interpreted languages in general, and Python in particular, are undeniably a useful tool for the modern programmer.

Python's built-in support for a wide range of data structures and its extensive standard libraries, as well as its high-level features like automatic reference-counted memory management, first-class functions, and powerful introspection allow Python code to be rapidly developed and easily maintained. This applies both to the development of Lucille itself and of applications using the library.

Unfortunately for users of Lucene, Java is an extremely insular environment[3] — the only outlet (or inlet) to it is the Java Native Interface [5], which is notoriously unwieldy

[2] `>>> import this`

[3] Java programmers are apparently content to re-implement the universe in Java.

and not widely used. Would-be users of Lucene are left with two options: go Java all the way, or run their Lucene code in a separate process (along with all the tedium and overhead which that entails, like interprocess communication and serialisation).

1.1.3 Lucille

Lucille was initially a more-or-less direct translation from Java to Python, however because of the above-mentioned language differences, this approach is not necessarily the most appropriate.

Lucille can take advantage of Python features where the equivalent is difficult (if not impossible) in Java. For example, generator functions in Python are a kind of co-routine (see [6, pp. 193–200]), allowing the concise expression of functions which operate on a sequence of input. State is encoded implicitly in the point of execution at which the generator was suspended. A generator function can easily be written in a lazy style, evaluating each element of its output sequence as needed. Java on the other hand has no equivalent language feature, and so the Java programmer must define classes whose instances explicitly keep track of their state, and design APIs to allow for such sequential evaluation. Lucille's implementation of token analysis already uses generators to great advantage.

Conversely, implementation details of the Python interpreter can lead to performance quirks in the Python port which would not otherwise be encountered in Java. For example, method calls are more expensive in Python than in Java, because the method's definition must be looked up dynamically, as opposed to being determined statically as in Java.

1.1.4 Other ports of Lucene

Lupy (http://divmod.org/projects/lupy) was a port of a very early release of Lucene to Python. Lupy was abandoned in May 2004. The Lucene codebase has evolved substantially since then, and so Lupy code cannot reasonably be reused in Lucille.

Lucy (http://lucene.apache.org/lucy/) began as a subproject of Lucene in June 2006, with the aim of implementing core Lucene functionality in C, which ports of Lucene to other high-level languages could build upon. At present no ported code has been released by the Lucy project.

PyLucene (http://pylucene.osafoundation.org/) provides a wrapper around a GCJ-compiled version of Lucene, allowing it to be accessed from Python. GCJ is still incomplete and unstable however, and as a result PyLucene is difficult to build

and not suited for production use.

Kinosearch (http://www.rectangular.com/kinosearch/) and Ferret (http://ferret.davebalmain.com/) are loose ports of Lucene to Perl and Ruby respectively. These are dynamic, interpreted languages like Python, and therefore some of the same challenges are faced when porting to these languages. The solutions adopted by both projects are to move an increasing proportion of their code to C extension modules, and in some cases making substantial changes to the internal and external interfaces of the library to better accommodate the performance quirks of the target language.

1.2 Performance of Lucille

The original version of Lucille is substantially slower than Lucene. Table 1.1 gives a summary of the average time taken to perform a range of common operations using Lucene, and the equivalent timings from Lucille. The corpus of documents used is the standard 20 Newsgroups testing corpus [7].

Table 1.1: Average time taken to perform common operations against a testing corpus

Operation	Lucene 2.1	Original Lucille
Boolean query (milliseconds)		
1 required term	1.545 (σ = 2.103)	24.868 (σ = 0.224)
1 required, 1 optional term	1.897 (σ = 3.547)	44.513 (σ = 0.444)
1 required, 1 prohibited term	1.783 (σ = 1.072)	46.162 (σ = 0.620)
2 required, 1 optional term	0.265 (σ = 0.592)	49.699 (σ = 0.683)
1 required, 2 optional terms	2.546 (σ = 0.926)	71.054 (σ = 0.795)
2 optional, 1 prohibited term	2.740 (σ = 1.182)	100.348 (σ = 0.802)
Analysis (tokens/second)	206,811	14,048

All timing and profiling results presented in this report were performed on Linux 2.6.24 with Python 2.5.2 and Sun Java 1.6.0_05, running on an AMD Phenom 9500 quad-core CPU with 2 GB of physical memory.

Timing tests were designed to isolate the operation in question as much as possible: measured times exclude interpreter/JVM startup and cleanup, as well as any initialisation required to run the timing tests.

It is worth noting, however, that Java's garbage collector makes no guarantees about when objects are destroyed, and so it is possible that garbage collection time is underrepresented in timing results for Lucene. This is in contrast to Python's reference counting scheme, which means that objects are always destroyed as soon as they are no longer referenced;[4] object destruction time will therefore always be included in timing for Lucille.

Times given are averaged across a large number of consecutive repetitions of the operation wherever appropriate. To minimise the impact of disk activity, an untimed run is performed immediately before the real run, to "warm" the operating system disk cache. All timings were performed on an otherwise-unloaded system, to avoid interference from other processes executing simultaneously.

1.3 Outline of this report

The subsequent chapters of this report detail each of the optimisation techniques employed during the project. The method and tools used are described, and timing results for the optimised version of the code are presented. The effectiveness of each technique is also discussed, as well as any limitations which may apply.

In the conclusion a summary of the project's results is given, along with a discussion of future directions for further improving the performance of Lucille.

Appendix A is a listing of all code discussed in this report. The full source of the original version of Lucille, as well as the scripts and data used to collect timing results for this report, are available on the attached CD-ROM and at `http://www.djc.id.au/2008/thesis/`.

[4] Objects participating in reference cycles are the exception to this rule, however Lucille code was written to deliberately avoid reference cycles for this reason.

Chapter 2

Reading from the index

In the Lucene `store` module, the `IndexInput` class and its subclasses provide the lowest layer of I/O abstraction for reading index data from storage. These classes are responsible for decoding raw bytes in persistent storage into integer and string data types, according to a number of encoding schemes described in [8]. Bytes are read sequentially from the current file position; seeking to a given file position is also supported.

The design of the `IndexInput` class hierarchy uses polymorphism to hide from calling code the details of where bytes are read from. `IndexInput` itself defines methods for decoding (`readVInt`, `readString`, etc) in terms of the `readByte` and `readBytes` methods. This allows subclasses to customise how data is read simply by overriding these two methods, while other classes can call the decoding methods to decode primitive data types without concern for how the data is read from storage. An equivalent strategy was adopted for Lucille while porting. Figure 2.2(a) illustrates the class as it exists in the original version of Lucille.

This design means, however, that the `read_byte` method of the `IndexInput` subclass in use is called once for nearly *every byte* read from the index. In Python, method calls are a notoriously costly operation, because each time the interpreter must not only look up the attribute in question, but then create an anonymous bound method object and call into it, which involves creating a new interpreter stack frame and several C-level calls. In contrast, Java methods can be looked up statically by the JIT compiler and called into at roughly the same speed as native functions.

The decoding algorithms, although quite simple, involve many integer arithmetic operations, such as left-shit and addition, for decoding each integer or string read from storage. Again due to Python's dynamic nature, the interpreter must look up the type of each variable for each arithmetic operation that is performed; although not as ex-

pensive as a method call, these type lookups make arithmetic operations in Python several orders of magnitude slower than the equivalent operation executed directly in compiled code would be. Java's JIT compiler, on the other hand, is able to use static type information to reduce such arithmetic operations to native CPU instructions.

The impact of these issues is evident in Figure 2.1, which shows the profiling results of a typical search operation.

Figure 2.1: `cProfile` output for a typical search operation, ordered by total function time

```
ncalls  tottime  percall  cumtime  percall filename:lineno(function)
607488    2.716    0.000    4.546    0.000 store.py:112(read_byte)
213353    1.668    0.000    2.759    0.000 /usr/lib64/python2.5/StringIO
    .py:208(write)
 55518    1.500    0.000    7.038    0.000 store.py:144(read_chars)
 55293    1.394    0.000   12.823    0.000 index.py:655(advance)
366794    1.281    0.000    4.235    0.000 store.py:132(read_vint)
607490    1.120    0.000    1.120    0.000 {method 'read' of 'file'
    objects}
607488    0.710    0.000    0.710    0.000 {ord}
213353    0.374    0.000    0.374    0.000 {unichr}
269624    0.345    0.000    0.345    0.000 {isinstance}
269602    0.333    0.000    0.333    0.000 {len}
 55518    0.319    0.000    0.461    0.000 /usr/lib64/python2.5/StringIO
     .py:54(__init__)
221002    0.297    0.000    0.297    0.000 {method 'append' of 'list'
    objects}
 55518    0.283    0.000    7.920    0.000 store.py:173(read_string)
   600    0.276    0.000    0.276    0.000 index.py:734(_index_offset)
213353    0.270    0.000    0.270    0.000 /usr/lib64/python2.5/StringIO
    .py:38(_complain_ifclosed)
 55518    0.234    0.000    0.354    0.000 /usr/lib64/python2.5/StringIO
     .py:258(getvalue)
...
```

2.1 Writing an extension module in C

The most effective way of reducing interpreter overhead is simply to eliminate the interpreter — that is, by re-writing the code in question in C as an extension module.

When issuing an `import` statement, the Python interpreter[1] is capable of loading the requested module from a dynamic shared object (DSO) using the `dlopen()` POSIX function or equivalent functionality on other platforms, in addition to the ordinary behaviour of loading the module from a `.py` file. In fact, a large number of the standard Python library modules are implemented as DSOs, although the `import` mechanism is deliberately designed to make this entirely transparent to the user. Modules implemented as DSOs are typically referred to as *extension modules*, since they "extend" the interpreter with native executable code.

The Python interpreter expects a specially-named initialisation function to be exported in the DSO for an extension module. When the extension module is first loaded into the interpreter,[2] this initialisation function is called. The function can then use Python API calls to initialise and populate the module with type definitions or other Python objects, which can then be accessed by Python code running in the interpreter.

The API provides functions for constructing and manipulating the basic Python builtin data types (`str`, `unicode`, `list`, `dict`, `tuple`, and others), as well as generic object manipulation operations (attribute lookup, rich comparison, arithmetic operations, calling Python callable objects, etc). Functions in the extension module can use the Python API to interact with other objects in the interpreter.

An excellent introduction to writing extension modules is in [9], and a full reference for the API is in [10].

To improve the performance of the `IndexInput` class, an extension module named `_store` was developed. This module provides a class named `BaseIndexInput`, which implements all of the decoding methods (`read_int`, `read_vint`, etc) as C functions operating directly with native data types. This avoids the overhead of the Python interpreter for these arithmetic operations. Like the original Python versions, these functions call the `read_byte` and `read_bytes` methods of themselves by using the Python API, in order to read raw data. This means that the `BaseIndexInput` class is not directly usable, since it does not itself provide a `read_byte` method for reading data. The `IndexInput` class in the Python `store` module thus no longer needs to implement these decoding methods, inheriting them instead from `BaseIndexInput`. It implements only the `read_byte` and `read_bytes` methods, which read data from a file. In this way the `IndexInput`

[1] Some prefer the term "the CPython interpreter", i.e. the original Python implementation written in C by Guido van Rossum and others. This is to distinguish it from Jython and other implementations of the Python interpreter, which do not provide a C API.

[2] A module may be *imported* many times, but the interpreter ensures that it is only *loaded* once. The same applies to modules implemented as Python code.

presents an unchanged interface to the rest of the code, but calls to its decoding methods are dispatched to compiled code which can perform arithmetic operations faster. Subclasses of `IndexInput` also benefit in the same way, by virtue of their inheritance, with no changes necessary to their code.

Figure 2.2(b) illustrates the new class structure after introducing `BaseIndexInput`. Table 2.1 shows the timing results after introducing `BaseIndexInput`. Listing A.1 shows the complete extension module.

Table 2.1: Search times with `BaseIndexInput` implemented in C

Operation	**Time (milliseconds)**
Boolean query	
1 required term	21.596 (σ = 0.195)
1 required, 1 optional term	36.903 (σ = 0.292)
1 required, 1 prohibited term	38.733 (σ = 0.349)
2 required, 1 optional term	32.526 (σ = 0.202)
1 required, 2 optional terms	54.776 (σ = 0.633)
2 optional, 1 prohibited term	82.206 (σ = 0.821)

2.2 Using `mmap`

The `mmap()` function is a standard POSIX system call which causes the contents of a file to be mapped into the address space of a process [11]. This allows the program to access the contents of the file simply by referring to locations in memory, rather than by issuing `seek()` and `read()` system calls. The virtual memory system is used to translate these memory accesses to buffers holding the file contents, and although it results in a page fault whenever unbuffered file contents are accessed, this is typically much more efficient than the system call and buffer copying overhead associated with repeated calls to `read()`. Some operating systems will also attempt to predict access patterns and read ahead accordingly, so that buffers are filled earlier.

The Win32 API provides functionality similar to `mmap()` with its `CreateFileMapping()` and `MapViewOfFile()` functions.

In an attempt to further improve the performance of reading index data from ordinary files (the most common case), the `_store` extension module was extended with another class, `MMapIndexInput`. This class provides the same interface as

`IndexInput`, and so can be used without modification by code which expects an `IndexInput` instance, but it exists apart from the inheritance hierarchy of `IndexInput`. Rather than directing all read operations through its `read_byte` and `read_bytes` methods, it uses memory-mapped files to access the raw bytes directly at the C level. Figure 2.2(c) shows the resulting `MMapIndexInput` class.

The `MMapIndexInput` class makes use of the standard Python `mmap` module to avoid having to worry about operating system specifics.

The `MMapIndexInput` class has a number of important limitations. If the operating system does not support memory-mapped files, or if the index files to be read are unable to be mapped (e.g. if they reside on a network file system which does not support memory-mapped access), the `mmap` module will fail to create the mapping and `MMapIndexInput` will not be usable. It is also not suitable for reading from large files on 32-bit platforms, where virtual address space is typically limited to 1–3 GB (and it is not uncommon for indexes to be much larger than this).

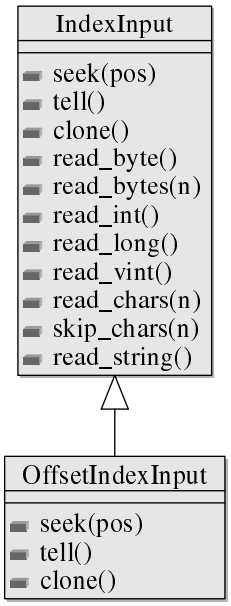
Timing results for `MMapIndexInput` are shown in Table 2.2. Figure 2.3 shows the updated profiling output using `MMapIndexInput`.

Table 2.2: Search times using `MMapIndexInput`

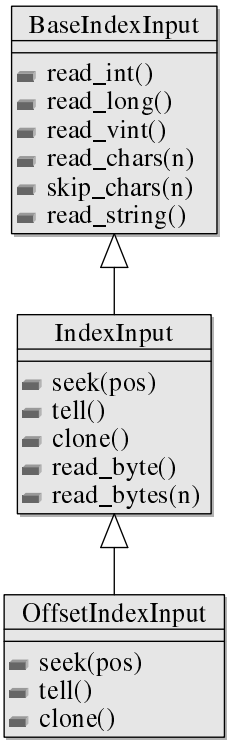
Operation	**Time (milliseconds)**
Boolean query	
1 required term	15.796 (σ = 0.312)
1 required, 1 optional term	28.296 (σ = 0.474)
1 required, 1 prohibited term	29.623 (σ = 0.417)
2 required, 1 optional term	19.095 (σ = 0.395)
1 required, 2 optional terms	38.270 (σ = 0.547)
2 optional, 1 prohibited term	63.874 (σ = 1.146)

Figure 2.2: Class diagrams showing the evolution of `IndexInput`

(a) The original `IndexInput` class with a subclass



(b) A stripped-down `IndexInput` class inheriting from the `BaseIndexInput` class



(c) The `MMapIndexInput` class provides the same interface as `IndexInput`, but with a faster implementation

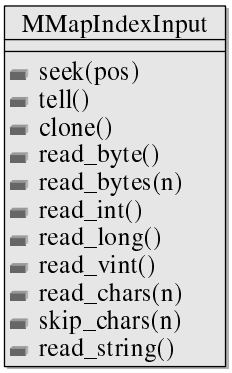


Figure 2.3: `cProfile` output using `MMapIndexInput`

```
ncalls  tottime  percall  cumtime  percall filename:lineno(function)
 55293    1.118    0.000    1.776    0.000 index.py:634(advance)
311276    0.373    0.000    0.373    0.000 {method 'read_vint' of '
    lucille._store.MMapIndexInput' objects}
   600    0.282    0.000    0.282    0.000 index.py:713(_index_offset)
   600    0.189    0.000    1.877    0.003 index.py:662(scan_to)
 55291    0.143    0.000    0.143    0.000 index.py:462(field_name)
 55295    0.101    0.000    0.101    0.000 index.py:585(__init__)
 55518    0.069    0.000    0.069    0.000 {method 'read_string' of '
    lucille._store.MMapIndexInput' objects}
   350    0.060    0.000    0.136    0.000 index.py:823(skip_to)
  6100    0.032    0.000    0.041    0.000 index.py:801(_do_advance)
  6100    0.018    0.000    0.058    0.000 index.py:813(advance)
     2    0.015    0.008    0.127    0.064 index.py:684(__init__)
  2493    0.014    0.000    0.103    0.000 index.py:630(__iter__)
   600    0.011    0.000    2.205    0.004 index.py:737(get)
  7649    0.009    0.000    0.009    0.000 {method 'append' of 'list'
     objects}
...
```

Chapter 3

Psyco

Psyco is an optimisation tool for Python based on the concept of "just-in-time specialisation" [12]. One of the most serious hurdles faced in runtime optimisation of a dynamic language like Python is that the type of each variable in a section of code is not known, and cannot easily be inferred, until immediately before it is executed. Psyco addresses this issue by automatically generating a new specialised machine code version of a block of Python code each time it is entered with different variable types. Due to the high initial compilation cost, and the potentially large memory requirements of compiling multiple versions of a function, Psyco provides a facility to specialise only those functions which are determined at runtime to be called most frequently or run longest.

A significant limitation of Psyco is that it is only capable of targeting the x86 architecture, and there are no plans to extend it to support other CPU architectures. There are also some limitations on the Python constructs which can be compiled by Psyco: most importantly, generators (using the `yield` statement) and nested functions will not be candidates for compilation.

Applying Psyco to a Python program is trivial. Figure 3.1 shows the changes applied to the benchmarking script in order to use Psyco with Lucille.

The total speed improvement from applying Psyco to a Python program depends on the characteristics of the program. The most substantial improvements arise when it is applied to code which performs mainly arithmetic operations on numeric data types. Psyco works within function boundaries, thus the overhead of function calls in Python is never eliminated. The most interesting optimisations can only be applied when Psyco is specialising for types which it "knows" about, such as `int` and `float`, as well as sequence types.

Figure 3.1: Applying Psyco to the benchmarking script

```
--- bench.py
+++ bench.py    (with Psyco)
@@ -60,4 +60,6 @@
     if len(sys.argv) != 2:
         sys.stderr.write('Usage: bench.py <index>\n')
         sys.exit(1)
+    import psyco
+    psyco.full()
     test(sys.argv[1])
```

As the timing results in Table 3.1 show,[1] applying Psyco to Lucille yields an overall improvement in speed of approximately 49%. Given the amount of effort required to achieve this, it is a very pleasing result. However the improvement is still much less substantial than other users of Psyco have reported. A likely cause is that the `search` module operates on Python class instances inside a number of tight loops when searching and scoring results, which means that even when Psyco is able to compile the code it is limited in the optimisations which it can perform.

Table 3.1: Search timing results with and without Psyco (using `MMapIndexInput`)

Operation	Original	Using Psyco
Boolean query (milliseconds)		
1 required term	15.374 (σ = 0.446)	9.394 (σ = 0.312)
1 required, 1 optional term	26.900 (σ = 0.252)	12.738 (σ = 0.267)
1 required, 1 prohibited term	28.375 (σ = 0.269)	12.967 (σ = 0.424)
2 required, 1 optional term	18.859 (σ = 0.325)	10.175 (σ = 0.178)
1 required, 2 optional terms	37.061 (σ = 0.290)	17.515 (σ = 0.269)
2 optional, 1 prohibited term	62.350 (σ = 0.524)	31.793 (σ = 0.407)

[1] Note that these timings are not directly comparable with other timing results presented in this report, because these results were collected from a 32-bit Python installation for compatibility with Psyco, whereas all other results are from a 64-bit installation.

Chapter 4

Lexical scanning with RE2C

The `analysis` module of Lucille, like its analogue in Lucene, provides a number of *tokenizers*, which convert text into a stream of tokens, and *token filters*, which perform some manipulation on the tokens in a stream. For example, `whitespace_tokenizer` splits its input string into tokens separated by whitespace characters, and `lowercase_filter` converts each token in a stream to lower case.

A tokenizer and zero or more token filters are then chained together to form an *analyzer*, which is used during indexing to convert the text of a document into discrete terms to be stored in the index. The same chain of token filters is typically also applied to user queries, in order to arrive at a set of query terms which are consistent with the terms stored in the index.

An important analyzer provided by Lucene is `StandardAnalyzer`, which, as its name suggests, is intended as a general-purpose analyzer suitable for use by many applications which do not have more specialised needs. The tokenizer component of this analyzer, `StandardTokenizer`, is generated from a set of regular expression rules using the JavaCC scanner generator.[1] The tokenizer includes rules for identifying e-mail addresses, dotted acronyms (such as I.B.M.), product numbers such as ISBNs and similar alphanumeric identifiers, words with embedded apostrophes, and other words with surrounding punctuation and whitespace stripped.

When porting `StandardTokenizer` to Lucille (as `standard_tokenizer`), the PLY package was used to construct a lexical scanner from a set of regular expression rules, taken from the JavaCC definition of `StandardTokenizer` in Lucene.

[1] From Lucene 2.3 onwards, `StandardTokenizer` is instead generated by the JFlex scanner generator. See https://issues.apache.org/jira/browse/LUCENE-966.

4.1 PLY

PLY (http://www.dabeaz.com/ply/), short for Python Lex-Yacc, is a Python implementation of a lexical scanner generator and parser generator, in the style of lex/yacc.

PLY was originally designed as teaching tool, therefore greater emphasis is placed on informative error messages and not on execution efficiency. Unlike other lexical scanner generators, no intermediate code generation phase is used. Instead, when tokenizing PLY simply iterates across all regular expressions in the rule-set in order of decreasing length, and uses the builtin Python regular expression module (`re`) to test for a match against the remaining input. Unfortunately this results in particularly poor performance for rule-sets where many tokens share a common initial substring, as is the case for `standard_tokenizer`, because the same initial substrings are scanned repeatedly by the `re` module.

Figure 4.1 shows a snippet of the rule definitions in Python, from which PLY builds a Lexer object at runtime. The string formatting operator (`%`) is used here to include other regular expressions defined earlier in the module.

Figure 4.1: A snippet of the PLY rule definitions

```
@TOKEN(ur'%(t_ALPHANUM)s([.\-_]%(t_ALPHANUM)s)*@%(t_ALPHANUM)s'
        ur'([.\-]%(t_ALPHANUM)s)+' % locals())
def t_EMAIL(self, t): return t

@TOKEN(ur'%(t_ALPHANUM)s(\.%(t_ALPHANUM)s)+' % locals())
def t_HOST(self, t): return t

@TOKEN(ur'(%(ALPHA)s\.){2,}' % locals())
def t_ACRONYM(self, t): return t
```

4.2 RE2C

RE2C (http://re2c.org/), originally published in [13], is an open-source lexical scanner generator written in C. Like other similar tools, it accepts a set of regular expressions describing tokens to be matched, along with C code defining the action to be taken when a match is found. It acts a preprocessor of C source files; in the original source file a special comment beginning with `/*!re2c` is inserted, containing the regular expression rules and code to be executed when each token is matched.

Because RE2C is designed to be adapted to a wide variety of applications, it does not have any ability to read its input from some source. Instead, the pattern matcher works with pointers into a pre-existing buffer; calling code must provide a number of definitions to control its behaviour:

- `YYCTYPE` is defined to be the character type (`char`, `wchar_t`, etc) which the pattern matcher is to work with;
- `YYCURSOR` is declared as a `YYCTYPE*` variable, and must be initialised to point to the first character of the input string;
- `YYMARKER` must also be declared as a `YYCTYPE*` variable, for internal use by the pattern matcher for implementing look-ahead;
- `YYLIMIT` may also be defined, as a `YYCTYPE*` expression pointing to the last position in the input string, along with a macro `YYFILL`, which is called by the pattern matcher when `YYLIMIT` is reached (these two definitions are optional).

RE2C supports a number of strategies for generating pattern-matching code. By default, the generated code uses a series of (possibly nested) `if` and `goto` statements, as in the example output given in Figure 4.2. RE2C can also make use of "computed gotos", a GCC language extension. These are faster, at the cost of much greater executable size.

Figure 4.2: A snippet of the code generated by RE2C using nested `if`s

```
yy19:
        ++cursor;
        yych = *cursor;
yy20:
        if(yych <= 0x00BF) {
            if(yych <= '9') {
                if(yych <= ',') goto yy13;
                if(yych <= '.') goto yy21;
```

4.3 Implementing the replacement `standard_tokenizer`

In order to make use of RE2C for `standard_tokenizer`, an extension module named `analysis._standard` was developed. This module defines a class named `standard_tokenizer` which supports the Python iterator protocol. Like the PLY version of `standard_tokenizer`, it is constructed with a `unicode` object as its input buffer, and returns a new token each time the `next()` method is called. The `next()` method, implemented in C by the `StandardTokenizer_iternext`

function, uses RE2C to scan for the next matching token, and returns a Python-level `Token` object constructed from it.

The implementation of `standard_tokenizer` takes advantage of the fact that Python `unicode` objects are NUL-terminated, by defining a catch-all rule at the end of the rule-set with pattern `[^]`. This pattern matches characters which are ignored by the tokenizer (e.g. whitespace), but it also matches the NUL character, so the C action code associated with this rule can test whether the end of the input buffer has been reached yet. This allows `YYLIMIT` to be avoided, which would otherwise cause RE2C to check for the end of the buffer each time it advances the character position.

Table 4.1 shows the resulting timing for analysing the 20 Newsgroups corpus. Listing A.2 shows the complete extension module.

Table 4.1: Analysis time using PLY and RE2C

Operation	**Using PLY**	**Using RE2C**
Analysis (tokens/second)	14,048	204,063

Chapter 5

Python-level optimisations

A number of optimisations at the Python level are commonly suggested when code is found to be running unacceptably slowly (see for example [14, 15, 16]). These are summarised on the Python wiki at `http://wiki.python.org/moin/PythonSpeed/PerformanceTips`. For the most part, these suggestions consist of avoiding gotchas in the implementation of the Python interpreter: e.g. before Python 2.5, building a string using concatenation inside a `for` loop resulted in quadratic running time, therefore the `str.join` method was the preferred alternative. Experienced Python programmers become aware of such issues and know to avoid them when writing code; in the original version of Lucille these gotchas were avoided.

One suggested optimisation, however, would not normally be applied automatically by a Python programmer due to its ugliness and fragility: that is the avoidance of redundant attribute accesses inside loops.

The highly dynamic nature of Python — almost any name in the interpreter can be rebound to a new value at any time — provides a great deal of flexibility and power for the programmer, but on the flip-side it means that the interpreter must look up each name in full every time it is evaluated. Furthermore, attribute lookup in Python is a non-trivial operation; depending on how the attribute has been defined, it could involve a slot lookup, a lookup in an instance dict, lookups in class dicts for each type in the method resolution order, testing for the descriptor protocol, and possibly even execution of arbitrary Python code [17]. Figure 5.1 shows C-level profiling of the interpreter, illustrating the large proportion of execution time spent in `PyObject_GenericGetAttr`, the Python interpreter function which handles attribute lookups on instances of Python classes, and `lookdict_string`, which handles lookups of string keys in a dict (the

majority of which are likely to be associated with attribute lookups).

Figure 5.1: C-level profiling of the Python interpreter during a search operation

```
 %   cumulative   self              self     total
time   seconds   seconds    calls   s/call   s/call  name
22.00     10.44    10.44  9345988     0.00     0.00  PyEval_EvalFrameEx
10.64     15.49     5.05 266971088    0.00     0.00  lookdict_string
 6.64     18.64     3.15 94562477     0.00     0.00
     PyObject_GenericGetAttr
 5.75     21.37     2.73 41464984     0.00     0.00  try_rich_compare
 4.84     23.67     2.30 72690002     0.00     0.00
     PyUnicodeUCS2_Compare
 3.03     25.11     1.44 38027268     0.00     0.00  PyObject_Malloc
 2.99     26.53     1.42 26377935     0.00     0.00  listiter_next
 2.95     27.93     1.40 246313509    0.00     0.00  PyDict_GetItem
 2.76     29.24     1.31 85794157     0.00     0.00
     PyObject_RichCompare
 2.50     30.42     1.19 29489996     0.00     0.00
     PyUnicodeUCS2_Decode
...
```

Addressing this overhead has been much discussed in the Python community, but one common suggestion is that when a name is looked up repeatedly (e.g. inside a `for` loop), some speed may be gained by aliasing its value to a local variable. This takes advantage of an implementation detail of the Python bytecode interpreter: local variables in each scope are kept in an array and accessed by index (LOAD_FAST bytecode instruction [18]), which means that referring to local variables is considerably faster than other kinds of name lookup.

Of course this optimisation is only suitable when it is known that the name in question will not be rebound by other code during the execution of the function. It is also of no benefit if the name lookup never (or rarely) occurs more than once during execution of the function; in this case the extra local variable access will have a negative impact on speed.

A disadvantage of this technique is that it makes code more difficult to read and maintain. For this reason, the technique would normally only be applied judiciously, to functions which have been shown by profiling to contribute largely to execution time.

To test the impact of this optimisation, Lucille's `search` module was carefully examined for any attribute lookups which occurred more than once in a function body. These repeated lookups were replaced with a single lookup stored in a local variable.

The timing results in Table 5.1 show that these optimisations little positive effect (and in some cases, a negative effect!) on the execution speed of the `search` module. This is because the majority of attribute accesses which occur inside tight loops could not be aliased as described above, because the value of the attribute changes at each iteration. This structure could be considered an artefact of the Java origins of the code, where attribute lookups are a cheap operation, since they are computed statically and optimised by the JIT compiler.

Table 5.1: Search timing results with and without reduced attribute accesses (using `MMapIndexInput`)

Operation	**Original**	**Reduced attribute accesses**
Boolean query (milliseconds)		
1 required term	15.796 (σ = 0.312)	16.684 (σ = 0.331)
1 required, 1 optional term	28.296 (σ = 0.474)	26.335 (σ = 0.321)
1 required, 1 prohibited term	29.623 (σ = 0.417)	28.088 (σ = 0.353)
2 required, 1 optional term	19.095 (σ = 0.395)	18.061 (σ = 0.130)
1 required, 2 optional terms	38.270 (σ = 0.547)	36.658 (σ = 0.419)
2 optional, 1 prohibited term	63.874 (σ = 1.146)	61.351 (σ = 0.407)

Chapter 6

Search using CLucene and Boost.Python

6.1 CLucene

CLucene is a port of Lucene to C++. It is a direct port, that is, its structure adheres closely to Lucene's, for the most part retaining the same class names with the same interfaces.[1] CLucene development typically lags a number of versions behind the latest release of Lucene, and does not introduce any new features not found in Lucene.

This close adherence to the original makes CLucene ideal for integration into Lucille, which is also a direct port and can be expected to have (roughly) compatible internal interfaces.

Once the speed of index reading in Lucille was drastically improved (see Chapter 2), further profiling showed that methods in the search module were a major factor in execution time for searches. As an alternative to the rewrite-it-in-C approach, the existence of CLucene presented an opportunity to replace the poorly-performing Python version of the search module with a faster C++ implementation. Because CLucene and Lucille are both direct ports of Lucene, a replacement search module based on CLucene should ideally provide equivalent behaviour and search results as the existing Python version.

However Python's API is entirely C, and makes no provisions for integration with the higher-level features of C++. Writing Python interfaces for C++ objects by hand would be a tedious process, but the Boost.Python library provides a solution to this.

[1] The lack of automatic garbage collection complicates the CLucene code in a number of ways which do not apply to Lucene.

6.2 Boost.Python

Boost (http://www.boost.org) is a collection of open-source C++ libraries providing useful functionality which is commonly required in C++ projects. It is a proving ground for new libraries to be included in the C++ standard.

The Boost.Python library takes advantage of C++ template metaprogramming techniques to simplify the interaction of C++ code with the Python C API [19].

The original purpose of Boost.Python was to simplify the process of exposing C++ classes to the Python interpreter as Python classes. It is capable of exposing C++ methods, operators, and constructors, and default arguments and Python keyword arguments are supported. It includes features for automatically managing object references and ownership of C++ objects which have been passed into the Python interpreter, using a number of different call policies implemented as template functions.

Boost.Python also supports going in the opposite direction — that is, it provides simplified access to the object manipulation parts of the Python C API. At its core lies the Boost.Python `object` class, which manages a reference to a `PyObject*`, the basic building block of the Python interpreter. The `object` class provides methods for generic manipulation of Python objects, as well as overridden operators for arithmetic, comparison, key/index lookup, etc. These methods and operators are translated to Python C API calls on the underlying `PyObject*`, with appropriate error checking. Table 6.1 shows some examples of the use of the `object` class.

Table 6.1: Some examples of using the Boost.Python `object` class, and the equivalent Python API calls (without error checking)

Using Boost.Python	Using the Python API
`object o;`	`PyObject *o;`
`o.attr("doc")`	`PyObject_GetAttrString(o, "doc")`
`o += 4`	`PyNumber_InPlaceAdd(o, PyInt_FromLong(4))`
`extract<double>(o)`	`PyFloat_AsDouble(PyNumber_Float(o))`

This allows C++ code which interfaces with Python to be written much more simply than using the API directly. More importantly, it also automates the tedious and error-prone management of reference counts, which all code using the Python API must do.

For convenience Boost.Python also provides a number of specialised subclasses of the `object` class: `str`, `list`, `tuple`, `dict`, and `long_`. These classes correspond

to the most common builtin Python data types, and provide methods and operators which call the respective parts of the Python API directly, rather than the generic object manipulation API. Table 6.2 illustrates some examples of these derived `object` types.

Table 6.2: Some examples of using Boost.Python's derived `object` types

Using Boost.Python	Using the Python API
`object o, tuple t, str s;`	`PyObject *o, *t, *s;`
`o[2]`	`PyObject_GetItem(o, PyInt_FromLong(2))`
`t[2]`	`PyTuple_GET_ITEM(o, 2)`
`s % t`	`PyString_Format(s, t)`
`extract<char *>(s)`	`PyString_AS_STRING(s)`

Boost.Python also provides facilities for extracting native C++ data types (e.g. `int`, `std::string`) from convertible Python objects, translating C++ exceptions (which are not handled by the C-only interpreter) into Python exceptions, converting between Python and STL iterator types, and other features.

6.3 Integrating the CLucene search module

The search module of CLucene has only a small number of direct dependencies on classes in other modules, namely it uses the `IndexReader` class, and its related classes `Term`, `TermDocs`, and `TermEnum`, for accessing term data from the index. This means the module can be extracted from the CLucene code, along with some common utility modules, and used independently.

In addition to constructing a Python wrapper around the C++ search module, it was necessary to provide stub versions of these classes in C++ which allows the search module to call back into Python. The header files for these stub classes were cut back from their original versions, to remove methods which weren't required by the search module, and return types and parameter types were modified where necessary to work with Boost.Python objects rather than the original concrete CLucene types. A stub implementation was then produced for each of these classes, in which all method calls are simply translated to the appropriate method of an underlying Boost.Python object.

A number of changes were also made to the CLucene search module when integrating it: primarily the removal of unused references to CLucene header files, but also

some changes were necessary to account for slight interface differences between Lucille and CLucene.

Listing A.3 shows the implementation of these stub classes, along with a patch of the changes made to the CLucene search module to accommodate them, and the Boost.Python module for exporting to Python.

Table 6.3 shows the timing results after integrating the CLucene search module. The actual improvement in speed was much less than anticipated. This can be attributed to the overhead incurred at the boundary between Python and C++ code, compounded by the additional work which Boost.Python must do in correctly managing ownership of references to C++ objects and handling exceptions.

Table 6.3: Search timing results comparing original and CLucene-based search module (using `MMapIndexInput`)

Operation	Original	CLucene search
Boolean query (milliseconds)		
1 required term	15.796 (σ = 0.312)	10.145 (σ = 0.130)
1 required, 1 optional term	28.296 (σ = 0.474)	17.029 (σ = 0.146)
1 required, 1 prohibited term	29.623 (σ = 0.417)	15.215 (σ = 0.197)
2 required, 1 optional term	19.095 (σ = 0.395)	36.630 (σ = 0.570)
1 required, 2 optional terms	38.270 (σ = 0.547)	36.990 (σ = 0.653)
2 optional, 1 prohibited term	63.874 (σ = 1.146)	34.336 (σ = 0.533)

During implementation of the search module, many iterations were spent chasing segmentation faults arising when Python or C++ code was still referring to objects which the other had cleaned up. This demonstrates that correct handling of memory ownership and reference lifetime between Python and C++ is tricky, and that great care must be taken in spite of all the facilities provided by Boost.Python to ease this task.

6.4 Future directions

When testing the integration of the CLucene search module, a number of discrepancies in search results were observed. Future work could determine the cause of these discrepancies and iron out any differences between the original and CLucene search implementations which would cause results to differ.

Chapter 7

Conclusion

In the previous chapters a number of techniques for improving the performance of the Lucille search and indexing library are described. The results of these improvements are summarised in Table 7.1.

Table 7.1: Summary of timing results

Operation	Original	`BaseII`	`MMapII`	Psyco	CLucene
Boolean query (milliseconds)					
1 required term	24.868	21.596	15.796	9.394	10.145
1 required, 1 optional term	44.513	36.903	28.296	12.738	17.029
1 required, 1 prohibited term	46.162	38.733	29.623	12.967	15.215
2 required, 1 optional term	49.699	32.526	19.095	10.175	36.630
1 required, 2 optional terms	71.054	54.776	38.270	17.515	36.990
2 optional, 1 prohibited term	100.348	82.206	63.874	31.793	34.336

Operation	Original	Using RE2C
Analysis (tokens/second)	14,048	204,063

The traditional approach to optimising a piece of code in a high-level dynamic language, such as Python, is to move time-critical sections to a compiled language and write an interface back to the higher-level language. This represents a trade-off between all the benefits of the higher-level language on the one hand — clarity, brevity, ease of maintenance, rapid development — and on the other hand, speed. In the case of Python,

making these kinds of trade-offs is eased by the simple and clear Python C API, and by other tools like Boost.Python.

Some overhead is unavoidably incurred at the boundary between Python and other languages, both in terms of execution speed and development effort. The larger the gap between the two languages at this boundary, the greater its overhead: for instance, a greater overhead is incurred in wrapping C++ objects using Boost.Python, than directly implementing Python objects in C. Therefore an important goal when moving is to minimise such boundaries.

The replacement of PLY with RE2C, described in Chapter 4, in particular yielded an impressive 13× speedup in the `standard_analyzer` class with relatively little difficulty, demonstrating that careful choice of tools is just as important for performance as choice of algorithms and language.

The use of the Psyco "just-in-time specialiser", described in Chapter 3, yielded an even more impressive speed increase for even less effort, but the serious limitation of only supporting the x86 architecture rules its use in many projects.

References

[1] J. Roskind, G. van Rossum, and A. Rigo, *The Python profilers*, 1994. Available from http://docs.python.org/lib/profile.html

[2] G. Salton, A. Wong, and C. S. Yang, "A vector space model for automatic indexing," *Communications of the ACM*, vol. 18, no. 11, p. 613–620, 1975. Available from http://www.cs.uiuc.edu/class/fa05/cs511/Spring05/other_papers/p613-salton.pdf

[3] G. van Rossum, "Python 0.9.1," article <2963@charon.cwi.nl> in newsgroup alt.sources, 19 February 1991, available from http://ftp.funet.fi/pub/archive/alt.sources/volume91/Feb/910220.10.gz ff.

[4] S. Ferg, "Python and Java: a side-by-side comparison," 2007. Available from http://www.ferg.org/projects/python_java_side-by-side.html

[5] Sun Microsystems, *Java Native Interface*, 2006. Available from http://java.sun.com/javase/6/docs/technotes/guides/jni/

[6] D. E. Knuth, *The art of computer programming, volume 1: Fundamental algorithms*, 3rd ed. Addison-Wesley, 1997.

[7] K. Lang, "The 20 Newsgroups data set," 1996. Available from http://people.csail.mit.edu/jrennie/20Newsgroups/

[8] The Apache Software Foundation, *Apache Lucene index file formats*, 2008, version 2.1. Available from http://lucene.apache.org/java/2_1_0/fileformats.html

[9] G. van Rossum, *Extending and embedding the Python interpreter*, 2007, version 2.5.1. Available from http://www.python.org/doc/2.5.1/ext/

[10] G. van Rossum, *Python/C API reference manual*, 2007, version 2.5.1. Available from http://www.python.org/doc/2.5.1/api/

[11] *Standard for information technology — portable operating system interface (POSIX). System interfaces*, IEEE Std. 1003.1–2004, 2004. Available from http://ieeexplore.ieee.org/servlet/opac?punumber=9157

[12] A. Rigo, "Representation-based just-in-time specialization and the Psyco prototype for Python," in *PEPM '04: Proceedings of the 2004 ACM SIGPLAN symposium on partial evaluation and semantics-based program manipulation*. ACM Press, 2004, p. 15–26. Available from http://psyco.sourceforge.net/theory_psyco.pdf

[13] P. Bumbulis and D. D. Cowan, "RE2C: a more versatile scanner generator," *ACM letters on programming languages and systems*, vol. 2, no. 1–4, p. 70–84, March–December 1993. Available from http://citeseer.ist.psu.edu/bumbulis94rec.html

[14] S. Montanaro, "Python Enhancement Proposal 266: Optimizing global variable/attribute access," 2001. Available from http://www.python.org/dev/peps/pep-0266/

[15] D. Nečas, "Python optimization tips," 2004. Available from http://web.archive.org/web/20060716234857/http://trific.ath.cx/resources/python/optimization/

[16] G. van Rossum, "An optimization anecdote," 2002. Available from http://www.python.org/doc/essays/list2str/

[17] S. Chaturvedi, "Python attributes and methods," 2005. Available from http://www.cafepy.com/article/python_attributes_and_methods/

[18] G. van Rossum, *Python Library Reference*, 2007, version 2.5.1. Available from http://www.python.org/doc/2.5.1/lib/

[19] D. Abrahams and R. W. Grosse-Kunstleve, "Building hybrid systems with Boost.Python," 2003. Available from http://www.boost-consulting.com/writing/bpl.html

Appendix A

Code listings

A.1 `_store` extension module

`_storemodule.c`: C source file implementing the `_store` extension module

```
/* Copyright 2007 Dan Callaghan <djc@djc.id.au> */

/*
 * This file is part of Lucille, based on Apache Lucene
 * <http://lucene.apache.org>.
 *
 * Lucille is free software; you can redistribute it and/or modify it under the
 * terms of the GNU General Public License as published by the Free Software
 * Foundation; either version 3 of the License, or (at your option) any later
 * version.
 *
 * Lucille is distributed in the hope that it will be useful, but WITHOUT ANY
 * WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
 * FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
 * details.
 *
 * You should have received a copy of the GNU General Public License along with
 * this program. If not, see <http://www.gnu.org/licenses/>.
 */

#include <Python.h>
#include "structmember.h"


/* The following struct definition is copied verbatim from
```

```
     * Modules/mmapmodule.c in the Python source. This makes the code here fragile
     * (if Python update the mmap module, we have to update this) and means that
     * compiled versions of this extension may not be binary compatible across
30   * Python releases or even compiler versions.
     *
     * However it does let us get access to the mmap objects guts without any
     * overhead of going back through the Python interpreter.
     */

    #ifndef MS_WINDOWS
    #define UNIX
    #endif

40  typedef struct {
        PyObject_HEAD
        char *  data;
        size_t  size;
        size_t  pos;

    #ifdef MS_WINDOWS
        HANDLE  map_handle;
        HANDLE  file_handle;
        char *  tagname;
50  #endif

    #ifdef UNIX
        int fd;
    #endif

        int access;
    } mmap_object;

    #ifdef UNIX
60  #include <sys/mman.h>
    #endif

    /* End code copied from Modules/mmapmodule.c */


    static PyObject *threading_module = NULL;
    static PyObject *mmap_module = NULL;
    static PyObject *zero = NULL;
    static PyObject *four = NULL;
70  static PyObject *eight = NULL;
```

```
static PyObject *str_read_bytes = NULL;
static PyObject *str_read_byte = NULL;


#define BaseIndexInput_doc "filepos_lock should be acquired by callers around
    blocks which expect the \n" \
    "file position to remain consistent. "

#define SELF_READ_BYTE(in, in_obj, error) \
    if ((in_obj = PyObject_CallMethodObjArgs((PyObject *)self, str_read_byte, NULL)
        ) == NULL) \
        goto error; \
    if ((in = PyInt_AsLong(in_obj)) < 0) \
        goto error; \
    Py_DECREF(in_obj)

typedef struct {
    PyObject_HEAD
    PyObject *filepos_lock;
} BaseIndexInput;

static PyMemberDef BaseIndexInput_members[] = {
    {"filepos_lock", T_OBJECT_EX, offsetof(BaseIndexInput, filepos_lock), 0,
        "for callers, to hold around seek operations"},
    {NULL} /* sentinel */
};

static PyObject *BaseIndexInput_new(PyTypeObject *type, PyObject *args,
        PyObject *kwargs) {
    BaseIndexInput *self;

    if ((self = (BaseIndexInput *)type->tp_alloc(type, 0)) == NULL)
        goto error;
    self->filepos_lock = NULL;
    return (PyObject *)self;

error:
    return NULL;
}

static int BaseIndexInput_init(BaseIndexInput *self, PyObject *args, PyObject *
    kwargs) {
    /* accept any args but ignore them */
    /*static char *kwlist[] = {NULL};
```

```
        if (!PyArg_ParseTupleAndKeywords(args, kwargs, "Sk", kwlist,
                &filename, &self->length))
            goto error;*/

        if (!(self->filepos_lock = PyObject_CallMethod(threading_module, "RLock",
                NULL)))
            goto error;

120     return 0;

    error:
        return -1;
    }

    static void BaseIndexInput_dealloc(BaseIndexInput *self) {
        Py_XDECREF(self->filepos_lock);
        self->ob_type->tp_free((PyObject *)self);
    }
130
    static PyObject *BaseIndexInput_read_chars(BaseIndexInput *self, PyObject *args) {
        unsigned long n = -1;
        if (!PyArg_ParseTuple(args, "k", &n))
            return NULL;
        PyObject *retval = NULL;
        if (!(retval = PyUnicode_FromUnicode(NULL, n)))
            return NULL;
        Py_UNICODE *buff = PyUnicode_AS_UNICODE(retval);
        int i;
140     for (i = 0; i < n; i ++) {
            PyObject *in_obj;
            long in;
            SELF_READ_BYTE(in, in_obj, error);
            unsigned char b = (unsigned char) in;
            if ((b & 0x80) == 0) {
                buff[i] = b;
            } else if ((b & 0xE0) != 0xE0) {
                SELF_READ_BYTE(in, in_obj, error);
                unsigned char b_lo = (unsigned char) in;
150             buff[i] = (b & 0x1F) << 6 | (b_lo & 0x3F);
            } else {
                SELF_READ_BYTE(in, in_obj, error);
                unsigned char b_mid = (unsigned char) in;
                SELF_READ_BYTE(in, in_obj, error);
                unsigned char b_lo = (unsigned char) in;
```

```
                buff[i] = (b & 0x0F) << 12 | (b_mid & 0x3F) << 6 | (b_lo & 0x3F);
            }
        }
        return retval;

    error:
        Py_DECREF(retval);
        return NULL;
    }

    static PyObject *BaseIndexInput_skip_chars(BaseIndexInput *self, PyObject *args) {
        unsigned long n = -1;
        if (!PyArg_ParseTuple(args, "k", &n))
            return NULL;
        int i;
        for (i = 0; i < n; i ++) {
            PyObject *in_obj;
            long in;
            SELF_READ_BYTE(in, in_obj, error);
            unsigned char b = (unsigned char) in;
            if ((b & 0x80) == 0) {
                /* pass */
            } else if ((b & 0xE0) != 0xE0) {
                SELF_READ_BYTE(in, in_obj, error);
            } else {
                SELF_READ_BYTE(in, in_obj, error);
                SELF_READ_BYTE(in, in_obj, error);
            }
        }
        Py_RETURN_NONE;

    error:
        return NULL;
    }

    static PyObject *BaseIndexInput_read_int(BaseIndexInput *self, PyObject *_) {
            /* METH_NOARGS */
        PyObject *in_obj;
        if (!(in_obj = PyObject_CallMethodObjArgs((PyObject *)self, str_read_bytes,
                four, NULL)))
            return NULL;

        unsigned char *in;
        Py_ssize_t in_len;
```

```
200     PyString_AsStringAndSize(in_obj, &in, &in_len);
        if (in_len != 4) {
            Py_DECREF(in_obj);
            return PyErr_Format(PyExc_ValueError,
                    "read_bytes(4) returned string of length %zd", in_len);
        }

        long n = (long) (in[0] << 24 | in[1] << 16 | in[2] << 8 | in[3]);
        Py_DECREF(in_obj);
        return PyInt_FromLong(n);
210 }

    static PyObject *BaseIndexInput_read_long(BaseIndexInput *self, PyObject *_) {
            /* METH_NOARGS */
        PyObject *in_obj;
        if (!(in_obj = PyObject_CallMethodObjArgs((PyObject *)self, str_read_bytes,
                eight, NULL)))
            return NULL;

        unsigned char *in;
220     Py_ssize_t in_len;
        PyString_AsStringAndSize(in_obj, &in, &in_len);
        if (in_len != 8) {
            Py_DECREF(in_obj);
            return PyErr_Format(PyExc_ValueError,
                    "read_bytes(8) returned string of length %zd", in_len);
        }

        PY_LONG_LONG n = ((PY_LONG_LONG) in[0] << 56 |
                (PY_LONG_LONG) in[1] << 48 |
230             (PY_LONG_LONG) in[2] << 40 |
                (PY_LONG_LONG) in[3] << 32 |
                (PY_LONG_LONG) in[4] << 24 |
                (PY_LONG_LONG) in[5] << 16 |
                (PY_LONG_LONG) in[6] << 8 |
                in[7]);
        return PyLong_FromLongLong(n);
    }

    static PyObject *BaseIndexInput_read_vint(BaseIndexInput *self, PyObject *_) {
240         /* METH_NOARGS */
        PyObject *in_obj;
        long in;
        unsigned long long val = 0;
```

```
        unsigned int shift = 0;
        do {
            SELF_READ_BYTE(in, in_obj, error);
            val += ((unsigned char) in & 127) << shift;
            shift += 7;
        } while ((unsigned char) in & 128);
        return PyLong_FromUnsignedLongLong(val);

    error:
        return NULL;
    }

    static PyObject *BaseIndexInput_read_string(BaseIndexInput *self, PyObject *_) {
            /* METH_NOARGS */
        PyObject *retval = NULL;
        PyObject *in_obj;
        int in;

        /* read_vint */
        unsigned long long n = 0;
        unsigned int shift = 0;
        do {
            SELF_READ_BYTE(in, in_obj, error_no_retval);
            n += ((unsigned char) in & 127) << shift;
            shift += 7;
        } while ((unsigned char) in & 128);

        /* read_chars */
        if (!(retval = PyUnicode_FromUnicode(NULL, n)))
            return NULL;
        Py_UNICODE *buff = PyUnicode_AS_UNICODE(retval);
        int i;
        for (i = 0; i < n; i ++) {
            SELF_READ_BYTE(in, in_obj, error);
            unsigned char b = (unsigned char) in;
            if ((b & 0x80) == 0) {
                buff[i] = b;
            } else if ((b & 0xE0) != 0xE0) {
                SELF_READ_BYTE(in, in_obj, error);
                unsigned char b_lo = (unsigned char) in;
                buff[i] = (b & 0x1F) << 6 | (b_lo & 0x3F);
            } else {
                SELF_READ_BYTE(in, in_obj, error);
                unsigned char b_mid = (unsigned char) in;
```

```
            SELF_READ_BYTE(in, in_obj, error);
            unsigned char b_lo = (unsigned char) in;
290         buff[i] = (b & 0x0F) << 12 | (b_mid & 0x3F) << 6 | (b_lo & 0x3F);
        }
    }
    return retval;

error:
    Py_DECREF(retval);
error_no_retval:
    return NULL;
}
300
static PyMethodDef BaseIndexInput_methods[] = {
    {"read_chars", (PyCFunction) BaseIndexInput_read_chars, METH_VARARGS,
        "Reads the given number of characters as a unicode.\n"
        "\n"
        "Lucene uses Java's \"modified UTF-8\":\n"
        "<http://java.sun.com/j2se/1.5.0/docs/api/java/io/DataInput.html#modified-
            utf-8>."},
    {"skip_chars", (PyCFunction) BaseIndexInput_skip_chars, METH_VARARGS,
        "Like read_chars, but discards the characters read."},
    {"read_int", (PyCFunction) BaseIndexInput_read_int, METH_NOARGS,
310     "Reads a signed 32-bit integer value."},
    {"read_long", (PyCFunction) BaseIndexInput_read_long, METH_NOARGS,
        "Reads a signed 64-bit integer value."},
    {"read_vint", (PyCFunction) BaseIndexInput_read_vint, METH_NOARGS,
        "Reads a Lucene variable-length integer value.\n"
        "\n"
        "Lucene also defines #readVLong(), but read_vint can be used for both."},
    {"read_string", (PyCFunction) BaseIndexInput_read_string, METH_NOARGS,
        "Reads a Lucene variable-length string (unicode) value."},
    {NULL} /* sentinel */
320 };

static PyTypeObject BaseIndexInputType = {
    PyObject_HEAD_INIT(NULL)
    0,                          /* ob_size */
    "lucille._store.BaseIndexInput",/* tp_name */
    sizeof(BaseIndexInput),     /* tp_basicsize */
    0,                          /* tp_itemsize */
    (destructor) BaseIndexInput_dealloc, /* tp_dealloc */
    0,                          /* tp_print */
330 0,                          /* tp_getattr */
```

```
        0,                          /* tp_setattr */
        0,                          /* tp_compare */
        0,                          /* tp_repr */
        0,                          /* tp_as_number */
        0,                          /* tp_as_sequence */
        0,                          /* tp_as_mapping */
        0,                          /* tp_hash  */
        0,                          /* tp_call */
        0,                          /* tp_str */
340     0,                          /* tp_getattro */
        0,                          /* tp_setattro */
        0,                          /* tp_as_buffer */
        Py_TPFLAGS_DEFAULT | Py_TPFLAGS_BASETYPE, /* tp_flags */
        BaseIndexInput_doc,         /* tp_doc */
        0,                          /* tp_traverse */
        0,                          /* tp_clear */
        0,                          /* tp_richcompare */
        0,                          /* tp_weaklistoffset */
        0,                          /* tp_iter */
350     0,                          /* tp_iternext */
        BaseIndexInput_methods,     /* tp_methods */
        BaseIndexInput_members,     /* tp_members */
        0,                          /* tp_getset */
        0,                          /* tp_base */
        0,                          /* tp_dict */
        0,                          /* tp_descr_get */
        0,                          /* tp_descr_set */
        0,                          /* tp_dictoffset */
        (initproc) BaseIndexInput_init, /* tp_init */
360     0,                          /* tp_alloc */
        BaseIndexInput_new,         /* tp_new */
    };


    #define MMapIndexInput_doc "IndexInput implementation that uses mmap() for
        accessing files."

    typedef struct {
        PyObject_HEAD
        PyObject *filepos_lock;
370     mmap_object *mmap;
        Py_ssize_t filepos;
        Py_ssize_t offset;
        Py_ssize_t length; /* the entire file is always mapped */
```

```
} MMapIndexInput;

static PyMemberDef MMapIndexInput_members[] = {
    {"filepos_lock", T_OBJECT_EX, offsetof(MMapIndexInput, filepos_lock), RO,
        "for callers, to hold around seek operations"},
    {"mmap", T_OBJECT_EX, offsetof(MMapIndexInput, mmap), RO,
        "underlying mmap object"},
    /* XXX these should be T_PYSSIZET, but that is missing from Python 2.5 */
    {"offset", T_LONGLONG, offsetof(MMapIndexInput, offset), 0,
        "file position offset (for compound files)"},
    {"length", T_LONGLONG, offsetof(MMapIndexInput, length), 0,
        "valid file length (may be less than real file length, for compound files)"
            },
    {NULL} /* sentinel */
};

static PyObject *MMapIndexInput_new(PyTypeObject *type, PyObject *args,
        PyObject *kwargs) {
    MMapIndexInput *self;

    if ((self = (MMapIndexInput *)type->tp_alloc(type, 0)) == NULL)
        goto error;
    self->filepos_lock = NULL;
    self->mmap = NULL;
    return (PyObject *)self;

error:
    return NULL;
}

static int MMapIndexInput_init(MMapIndexInput *self, PyObject *args, PyObject *
    kwargs) {
    PyObject *file;
    Py_ssize_t length;
    static char *kwlist[] = {"file", "length", NULL};
    if (!PyArg_ParseTupleAndKeywords(args, kwargs, "On", kwlist,
            &file, &length))
        goto error;

    if (!(self->filepos_lock = PyObject_CallMethod(threading_module, "RLock",
            NULL)))
        goto error;

    PyObject *fd_obj;
```

```
        int fd;
        if (!(fd_obj = PyObject_CallMethod(file, "fileno", NULL)))
            goto error;
        fd = PyInt_AsLong(fd_obj);
420     Py_DECREF(fd_obj);
        if (fd == -1)
            goto error;

    #ifdef MS_WINDOWS
        PyObject *access_read;
        if (!(access_read = PyObject_GetAttrString(mmap_module, "ACCESS_READ")))
            goto error;
        self->mmap = (mmap_object *)PyObject_CallMethod(mmap_module, "mmap", "insO",
                fd, 0, NULL, access_read);
430     Py_DECREF(access_read);
    #else
        self->mmap = (mmap_object *)PyObject_CallMethod(mmap_module, "mmap", "inii",
                fd, 0, MAP_SHARED, PROT_READ);
    #endif
        if (!self->mmap)
            goto error;

        self->filepos = 0;
        self->offset = 0;
440     self->length = length;

        return 0;

    error:
        return -1;
    }

    static PyObject *MMapIndexInput_clone(MMapIndexInput *self, PyObject *_) {
            /* METH_NOARGS */
450     MMapIndexInput *clone = NULL;

        if (!(clone = (MMapIndexInput *)self->ob_type->tp_new(self->ob_type,
                NULL, NULL)))
            return NULL;

        if (!(clone->filepos_lock = PyObject_CallMethod(threading_module, "RLock",
                NULL)))
            goto error;
        Py_INCREF(self->mmap);
```

```
460     clone->mmap = self->mmap;
        clone->filepos = self->filepos;
        clone->offset = self->offset;
        clone->length = self->length;

        return (PyObject *)clone;

    error:
        Py_XDECREF(clone);
        return NULL;
470 }

    static void MMapIndexInput_dealloc(MMapIndexInput *self) {
        Py_XDECREF(self->filepos_lock);
        Py_XDECREF(self->mmap);
        self->ob_type->tp_free((PyObject *)self);
    }

    static PyObject *MMapIndexInput_tell(MMapIndexInput *self, PyObject *_) {
            /* METH_NOARGS */
480     return PyInt_FromSsize_t(self->filepos - self->offset);
    }

    static PyObject *MMapIndexInput_seek(MMapIndexInput *self, PyObject *args) {
        Py_ssize_t pos;
        if (!PyArg_ParseTuple(args, "n", &pos))
            return NULL;
        if (pos > self->length) {
            PyErr_SetString(PyExc_ValueError, "seek beyond end of mmap");
            return NULL;
490     }
        self->filepos = self->offset + pos;
        Py_RETURN_NONE;
    }

    static inline unsigned char MMapIndexInput__read_byte(MMapIndexInput *self) {
        return self->mmap->data[self->filepos ++];
    }

    static inline void MMapIndexInput__read_bytes(MMapIndexInput *self,
500         unsigned char **buff, Py_ssize_t n) {
        unsigned char *retval = (unsigned char *)self->mmap->data + self->filepos;
        self->filepos += n;
        *buff = retval;
```

```
}

static PyObject *MMapIndexInput_read_byte(MMapIndexInput *self, PyObject *_) {
        /* METH_NOARGS */
    return Py_BuildValue("B", MMapIndexInput__read_byte(self));
}

static PyObject *MMapIndexInput_read_bytes(MMapIndexInput *self, PyObject *args) {
    Py_ssize_t n;
    if (!PyArg_ParseTuple(args, "n", &n))
        return NULL;
    unsigned char *buff;
    MMapIndexInput__read_bytes(self, &buff, n);
    return Py_BuildValue("s#", buff, n);
}

static PyObject *MMapIndexInput_read_chars(MMapIndexInput *self, PyObject *args) {
    unsigned long n = -1;
    if (!PyArg_ParseTuple(args, "k", &n))
        return NULL;
    PyObject *retval = NULL;
    if (!(retval = PyUnicode_FromUnicode(NULL, n)))
        return NULL;
    Py_UNICODE *buff = PyUnicode_AS_UNICODE(retval);
    int i;
    for (i = 0; i < n; i ++) {
        unsigned char b = MMapIndexInput__read_byte(self);
        if ((b & 0x80) == 0) {
            buff[i] = b;
        } else if ((b & 0xE0) != 0xE0) {
            unsigned char b_lo = MMapIndexInput__read_byte(self);
            buff[i] = (b & 0x1F) << 6 | (b_lo & 0x3F);
        } else {
            unsigned char b_mid = MMapIndexInput__read_byte(self);
            unsigned char b_lo = MMapIndexInput__read_byte(self);
            buff[i] = (b & 0x0F) << 12 | (b_mid & 0x3F) << 6 | (b_lo & 0x3F);
        }
    }
    return retval;
}

static PyObject *MMapIndexInput_skip_chars(MMapIndexInput *self, PyObject *args) {
    unsigned long n = -1;
    if (!PyArg_ParseTuple(args, "k", &n))
```

```
            return NULL;
        int i;
        for (i = 0; i < n; i ++) {
            unsigned char b = MMapIndexInput__read_byte(self);
            if ((b & 0x80) == 0) {
                /* pass */
            } else if ((b & 0xE0) != 0xE0) {
                MMapIndexInput__read_byte(self);
            } else {
                MMapIndexInput__read_byte(self);
                MMapIndexInput__read_byte(self);
            }
        }
        Py_RETURN_NONE;
    }

    static PyObject *MMapIndexInput_read_int(MMapIndexInput *self, PyObject *_) {
            /* METH_NOARGS */
        unsigned char *in;
        MMapIndexInput__read_bytes(self, &in, 4);
        long n = (long) (in[0] << 24 | in[1] << 16 | in[2] << 8 | in[3]);
        return PyInt_FromLong(n);
    }

    static PyObject *MMapIndexInput_read_long(MMapIndexInput *self, PyObject *_) {
            /* METH_NOARGS */
        unsigned char *in;
        MMapIndexInput__read_bytes(self, &in, 8);
        PY_LONG_LONG n = ((PY_LONG_LONG) in[0] << 56 |
                (PY_LONG_LONG) in[1] << 48 |
                (PY_LONG_LONG) in[2] << 40 |
                (PY_LONG_LONG) in[3] << 32 |
                (PY_LONG_LONG) in[4] << 24 |
                (PY_LONG_LONG) in[5] << 16 |
                (PY_LONG_LONG) in[6] << 8 |
                in[7]);
        return PyLong_FromLongLong(n);
    }

    static PyObject *MMapIndexInput_read_vint(MMapIndexInput *self, PyObject *_) {
            /* METH_NOARGS */
        unsigned char in;
        unsigned long long val = 0;
        unsigned int shift = 0;
```

```
        do {
            in = MMapIndexInput__read_byte(self);
            val += (in & 127) << shift;
            shift += 7;
        } while (in & 128);
        return PyLong_FromUnsignedLongLong(val);
    }

600 static PyObject *MMapIndexInput_read_string(MMapIndexInput *self, PyObject *_) {
            /* METH_NOARGS */
        PyObject *retval = NULL;

        /* read_vint */
        unsigned char in;
        unsigned long long n = 0;
        unsigned int shift = 0;
        do {
            in = MMapIndexInput__read_byte(self);
610         n += (in & 127) << shift;
            shift += 7;
        } while (in & 128);

        /* read_chars */
        if (!(retval = PyUnicode_FromUnicode(NULL, n)))
            return NULL;
        Py_UNICODE *buff = PyUnicode_AS_UNICODE(retval);
        int i;
        for (i = 0; i < n; i ++) {
620         unsigned char b = MMapIndexInput__read_byte(self);
            if ((b & 0x80) == 0) {
                buff[i] = b;
            } else if ((b & 0xE0) != 0xE0) {
                unsigned char b_lo = MMapIndexInput__read_byte(self);
                buff[i] = (b & 0x1F) << 6 | (b_lo & 0x3F);
            } else {
                unsigned char b_mid = MMapIndexInput__read_byte(self);
                unsigned char b_lo = MMapIndexInput__read_byte(self);
                buff[i] = (b & 0x0F) << 12 | (b_mid & 0x3F) << 6 | (b_lo & 0x3F);
630         }
        }
        return retval;
    }

    static PyMethodDef MMapIndexInput_methods[] = {
```

```
        {"clone", (PyCFunction) MMapIndexInput_clone, METH_NOARGS, NULL},
        {"tell", (PyCFunction) MMapIndexInput_tell, METH_NOARGS, NULL},
        {"seek", (PyCFunction) MMapIndexInput_seek, METH_VARARGS, NULL},
        {"read_byte", (PyCFunction) MMapIndexInput_read_byte, METH_NOARGS, NULL},
640     {"read_bytes", (PyCFunction) MMapIndexInput_read_bytes, METH_VARARGS, NULL},
        {"read_chars", (PyCFunction) MMapIndexInput_read_chars, METH_VARARGS, NULL},
        {"skip_chars", (PyCFunction) MMapIndexInput_skip_chars, METH_VARARGS, NULL},
        {"read_int", (PyCFunction) MMapIndexInput_read_int, METH_NOARGS, NULL},
        {"read_long", (PyCFunction) MMapIndexInput_read_long, METH_NOARGS, NULL},
        {"read_vint", (PyCFunction) MMapIndexInput_read_vint, METH_NOARGS, NULL},
        {"read_string", (PyCFunction) MMapIndexInput_read_string, METH_NOARGS, NULL},
        {NULL} /* sentinel */
    };

650 static PyTypeObject MMapIndexInputType = {
        PyObject_HEAD_INIT(NULL)
        0,                          /* ob_size */
        "lucille._store.MMapIndexInput",/* tp_name */
        sizeof(MMapIndexInput),     /* tp_basicsize */
        0,                          /* tp_itemsize */
        (destructor) MMapIndexInput_dealloc, /* tp_dealloc */
        0,                          /* tp_print */
        0,                          /* tp_getattr */
        0,                          /* tp_setattr */
660     0,                          /* tp_compare */
        0,                          /* tp_repr */
        0,                          /* tp_as_number */
        0,                          /* tp_as_sequence */
        0,                          /* tp_as_mapping */
        0,                          /* tp_hash  */
        0,                          /* tp_call */
        0,                          /* tp_str */
        0,                          /* tp_getattro */
        0,                          /* tp_setattro */
670     0,                          /* tp_as_buffer */
        Py_TPFLAGS_DEFAULT | Py_TPFLAGS_BASETYPE, /* tp_flags */
        MMapIndexInput_doc,         /* tp_doc */
        0,                          /* tp_traverse */
        0,                          /* tp_clear */
        0,                          /* tp_richcompare */
        0,                          /* tp_weaklistoffset */
        0,                          /* tp_iter */
        0,                          /* tp_iternext */
        MMapIndexInput_methods,     /* tp_methods */
```

```
680     MMapIndexInput_members,     /* tp_members */
        0,                          /* tp_getset */
        0,                          /* tp_base */
        0,                          /* tp_dict */
        0,                          /* tp_descr_get */
        0,                          /* tp_descr_set */
        0,                          /* tp_dictoffset */
        (initproc) MMapIndexInput_init, /* tp_init */
        0,                          /* tp_alloc */
        MMapIndexInput_new,         /* tp_new */
690 };


    static PyMethodDef _store_methods[] = {
        {NULL}  /* sentinel */
    };

    PyMODINIT_FUNC init_store(void) {
        /* fetch ints */
        if (!(zero = PyInt_FromLong(0)))
700         goto error;
        if (!(four = PyInt_FromLong(4)))
            goto error;
        if (!(eight = PyInt_FromLong(8)))
            goto error;

        /* create strings */
        if (!(str_read_byte = PyString_FromString("read_byte")))
            goto error;
        if (!(str_read_bytes = PyString_FromString("read_bytes")))
710         goto error;

        /* import threading */
        PyObject *threading_module_name = NULL;
        if (!(threading_module_name = PyString_FromString("threading")))
            goto error;
        if (!(threading_module = PyImport_Import(threading_module_name)))
            goto error;
        Py_DECREF(threading_module_name);
        threading_module_name = NULL;
720
        /* import mmap */
        PyObject *mmap_module_name = NULL;
        if (!(mmap_module_name = PyString_FromString("mmap")))
```

```
            goto error;
        if (!(mmap_module = PyImport_Import(mmap_module_name)))
            goto error;
        Py_DECREF(mmap_module_name);
        mmap_module_name = NULL;

730     /* initialise this module */
        PyObject *m;

        BaseIndexInputType.tp_new = PyType_GenericNew;
        if (PyType_Ready(&BaseIndexInputType) < 0)
            goto error;
        MMapIndexInputType.tp_new = PyType_GenericNew;
        if (PyType_Ready(&MMapIndexInputType) < 0)
            goto error;

740     m = Py_InitModule3("lucille._store", _store_methods,
                "Provides native implementations of some classes in lucille.store.");

        Py_INCREF(&BaseIndexInputType);
        PyModule_AddObject(m, "BaseIndexInput", (PyObject *)&BaseIndexInputType);
        Py_INCREF(&MMapIndexInputType);
        PyModule_AddObject(m, "MMapIndexInput", (PyObject *)&MMapIndexInputType);

        return;

750 error:
        Py_XDECREF(str_read_byte);
        Py_XDECREF(str_read_bytes);
        Py_XDECREF(eight);
        Py_XDECREF(four);
        Py_XDECREF(zero);
        Py_XDECREF(mmap_module);
        Py_XDECREF(mmap_module_name);
        Py_XDECREF(threading_module);
        Py_XDECREF(threading_module_name);
760     return;
    }
```

Patch of changes to `store.py` to accomodate the `_store` extension module

```
Index: store.py
===================================================================
--- store.py    (revision 67)
+++ store.py    (revision 100)
```

```
@@ -22,6 +22,8 @@
 import threading
 from StringIO import StringIO

+from lucille._store import BaseIndexInput, MMapIndexInput
+
 # TODO locking

 """ Lucene Directory protocol <http://svn.apache.org/viewvc/lucene/java/tags/
     lucene_2_1_0/src/java/org/apache/lucene/store/Directory.java>
@@ -74,31 +76,33 @@

     def open_input(self, name):
         fullname = os.path.join(self.path, name)
-        return IndexInput(fullname, os.path.getsize(fullname))
+        try:
+            return MMapIndexInput(open(fullname, 'rb'), os.path.getsize(fullname))
+        except (EnvironmentError, OSError, IOError):
+            return IndexInput(fullname, os.path.getsize(fullname))


-class IndexInput(object):
+class IndexInput(BaseIndexInput):
     """ <http://svn.apache.org/viewvc/lucene/java/tags/lucene_2_1_0/src/java/org/
     apache/lucene/store/IndexInput.java>

     filepos_lock should be acquired by callers around blocks which expect the
     file position to remain consistent. """

-    __slots__ = ('filename', 'file', 'length', 'filepos_lock')
+    __slots__ = ('filename', 'file', 'length', 'offset')

-    def __init__(self, filename, length):
+    def __init__(self, filename, length, **kwargs):
+        super(IndexInput, self).__init__()
         self.filename = filename
         self.length = length
         self.file = open(self.filename, 'rb')
-        self.filepos_lock = threading.RLock()
-        self.seek(0) # for subclasses
+        self.offset = 0

-    def __len__(self): return self.length
-
```

```
         def seek(self, pos):
    -        self.file.seek(pos)
49  +        if pos > self.length: raise EOFError()
    +        self.file.seek(self.offset + pos)

         def tell(self):
    -        return self.file.tell()
    +        return self.file.tell() - self.offset

         def clone(self):
             """ Returns a new IndexInput which reads from the same file as this
    @@ -106,70 +110,12 @@
59           copied from this one. """
             p = self.tell()
             cl = self.__class__(self.filename, self.length)
    +        cl.offset = self.offset
             cl.seek(p)
             return cl

         def read_byte(self):
    -        # XXX could use mmap instead?
             return ord(self.file.read(1))
69
         def read_bytes(self, n):
             return self.file.read(n)
    -
    -    def read_int(self):
    -        n = (self.read_byte() << 24 | self.read_byte() << 16 |
    -                self.read_byte() << 8 | self.read_byte())
    -        if n > 0xefffffff:
    -            return n - 0xffffffff - 1
    -        else:
79  -            return n
    -
    -    def read_long(self):
    -        return (self.read_int() << 32 |
    -                self.read_byte() << 24 | self.read_byte() << 16 |
    -                self.read_byte() << 8 | self.read_byte())
    -
    -    def read_vint(self):
    -        """ Lucene also defines #readVLong(), but read_vint can be used for
    -        both. """
89  -        val = 0
    -        shift = 0
```

```
-        while True:
-            b = self.read_byte()
-            val += (b & 127) << shift
-            shift += 7
-            if not (b & 128):
-                return val
-
-    def read_chars(self, n):
 99 -        """ Reads n characters from the underlying file and returns them as a
    unicode object.
-
-        Lucene uses Java's "modified UTF-8": <http://java.sun.com/j2se/1.5.0/docs/
    api/java/io/DataInput.html#modified-utf-8>. """
-        buff = StringIO()
-        while n > 0:
-            b = self.read_byte()
-            if (b & 0x80) == 0:
-                buff.write(unichr(b & 0x7F))
-            elif (b & 0xE0) != 0xE0:
-                buff.write(unichr(((b & 0x1F) << 6) | (self.read_byte() & 0x3F)))
109 -            else:
-                buff.write(unichr(((b & 0x0F) << 12) | ((self.read_byte() & 0x3F)
    << 6) | (self.read_byte() & 0x3F)))
-            n -= 1
-        return buff.getvalue()
-
-    def skip_chars(self, n):
-        """ Like read_chars, but throws away the bytes. """
-        while n > 0:
-            b = self.read_byte()
-            if (b & 0x80) == 0:
119 -                pass
-            elif (b & 0xE0) != 0xE0:
-                self.read_byte()
-            else:
-                self.read_byte()
-                self.read_byte()
-            n -= 1
-
-    def read_string(self):
-        length = self.read_vint()
129 -        return self.read_chars(length)
```

A.2 `analysis._standard` extension module

`_standardmodule.re`: C source file (before preprocessing with RE2C) implementing `standard_tokenizer`

```
1
   /* Copyright 2007 Dan Callaghan <djc@djc.id.au> */

   /*
    * This file is part of Lucille, based on Apache Lucene
    * <http://lucene.apache.org>.
    *
    * Lucille is free software; you can redistribute it and/or modify it under the
    * terms of the GNU General Public License as published by the Free Software
    * Foundation; either version 3 of the License, or (at your option) any later
11  * version.
    *
    * Lucille is distributed in the hope that it will be useful, but WITHOUT ANY
    * WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
    * FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
    * details.
    *
    * You should have received a copy of the GNU General Public License along with
    * this program. If not, see <http://www.gnu.org/licenses/>.
    */
21
   #include <Python.h>
   #include "structmember.h"


   static PyObject *alphanum_type = NULL;
   static PyObject *apostrophe_type = NULL;
   static PyObject *acronym_type = NULL;
   static PyObject *company_type = NULL;
   static PyObject *email_type = NULL;
31 static PyObject *host_type = NULL;
   static PyObject *num_type = NULL;
   static PyObject *cj_type = NULL;
   static PyObject *analysis_module = NULL;


   #define StandardTokenizer_doc "<http://svn.apache.org/viewvc/lucene/java/tags/
       lucene_2_1_0/src/java/org/apache/lucene/analysis/standard/StandardTokenizer.jj>
       "
```

```
typedef struct {
    PyObject_HEAD
    PyUnicodeObject *input;
    Py_UNICODE *cursor;
    Py_UNICODE *limit;
} StandardTokenizer;

static PyMemberDef StandardTokenizer_members[] = {
    /*{"input", T_OBJECT_EX, offsetof(StandardTokenizer, input), RO,
        "input unicode"}, */
    {NULL} /* sentinel */
};

static PyObject *StandardTokenizer_new(PyTypeObject *type, PyObject *args,
        PyObject *kwargs) {
    StandardTokenizer *self;

    if ((self = (StandardTokenizer *)type->tp_alloc(type, 0)) == NULL)
        goto error;
    self->input = NULL;
    self->cursor = NULL;
    self->limit = NULL;
    return (PyObject *)self;

error:
    return NULL;
}

static int StandardTokenizer_init(StandardTokenizer *self, PyObject *args,
        PyObject *kwargs) {
    PyUnicodeObject *input;

    static char *kwlist[] = {"input", NULL};
    /* XXX should coerce to unicode? or at least accept str ... */
    if (!PyArg_ParseTupleAndKeywords(args, kwargs, "U", kwlist,
            &input))
        goto error;

    Py_INCREF(input);
    Py_XDECREF(self->input);
    self->input = input;

    self->cursor = PyUnicode_AS_UNICODE(self->input);
```

```
        /* XXX NOT We position self->limit one *beyond* the end of the string, because
         * Python always keeps a NUL terminator at the end of the buffer and the
         * re2c-generated code relies on this to detect the end of the buffer. */
        self->limit = self->cursor + PyUnicode_GET_SIZE(self->input);

        return 0;

    error:
91      return -1;
    }

    static void StandardTokenizer_dealloc(StandardTokenizer *self) {
        Py_XDECREF(self->input);
        self->ob_type->tp_free((PyObject *)self);
    }

    static PyObject *StandardTokenizer_iter(StandardTokenizer *self) {
        Py_INCREF(self);
101     return (PyObject *)self; /* per iterator proto */
    }

    static PyObject *StandardTokenizer_iternext(StandardTokenizer *self) {

        if (self->cursor < self->limit) {

    #define END_TOKEN(_type) type = _type; break;

            Py_UNICODE *cursor = self->cursor;
111         Py_UNICODE *marker = NULL;
            PyObject *type = NULL;
            for (;;) {
    /*!re2c
        re2c:define:YYCTYPE  = Py_UNICODE;
        re2c:define:YYCURSOR = cursor;
        re2c:define:YYMARKER = marker;
        re2c:yyfill:enable   = 0;
        re2c:indent:top      = 2;

121     LETTER = [\u0041-\u005a\u0061-\u007a\u00c0-\u00d6\u00d8-\u00f6\u00f8-\u00ff\
            u0100-\u1fff\uffa0-\uffdc];
        DIGIT = [\u0030-\u0039\u0660-\u0669\u06f0-\u06f9\u0966-\u096f\u09e6-\u09ef\
            u0a66-\u0a6f\u0ae6-\u0aef\u0b66-\u0b6f\u0be7-\u0bef\u0c66-\u0c6f\u0ce6-\
            u0cef\u0d66-\u0d6f\u0e50-\u0e59\u0ed0-\u0ed9\u1040-\u1049];
```

```
        KOREAN = [\uac00-\ud7af\u1100-\u11ff];
        ALPHA = LETTER+;
        P = [-_/.,];
        HAS_DIGIT = (LETTER|DIGIT)*DIGIT(LETTER|DIGIT)*;

        ALPHANUM = (LETTER|DIGIT|KOREAN)+;
        APOSTROPHE = ALPHA ("'" ALPHA)+;
        ACRONYM = ALPHA "." (ALPHA ".")+;
131     COMPANY = ALPHA [&@] ALPHA;
        EMAIL = ALPHANUM ([-._] ALPHANUM)* "@" ALPHANUM ([-.] ALPHANUM)+;
        HOST = ALPHANUM ("." ALPHANUM)+;
        NUM = (ALPHANUM P HAS_DIGIT
            | HAS_DIGIT P ALPHANUM
            | ALPHANUM (P HAS_DIGIT P ALPHANUM)+
            | HAS_DIGIT (P ALPHANUM P HAS_DIGIT)+
            | ALPHANUM P HAS_DIGIT (P ALPHANUM P HAS_DIGIT)+
            | HAS_DIGIT P ALPHANUM (P HAS_DIGIT P ALPHANUM)+
            );
141     CJ = [\u3040-\u318f\u3100-\u312f\u3040-\u309F\u30A0-\u30FF\u31F0-\u31FF\u3300-\
            u337f\u3400-\u4dbf\u4e00-\u9fff\uf900-\ufaff\uff65-\uff9f];

        ALPHANUM { END_TOKEN(alphanum_type); }
        APOSTROPHE { END_TOKEN(apostrophe_type); }
        ACRONYM { END_TOKEN(acronym_type); }
        COMPANY { END_TOKEN(company_type); }
        EMAIL { END_TOKEN(email_type); }
        HOST { END_TOKEN(host_type); }
        NUM { END_TOKEN(num_type); }
        CJ { END_TOKEN(cj_type); }
151     [^] {
            // discard everything else
            self->cursor = cursor;
            if (self->cursor < self->limit)
                continue;
            else
                return NULL; // exhausted
        }
    */
            }
161
            /* create and return a new Token */
            PyObject *token = NULL;
            PyObject *u = PyUnicode_FromUnicode(self->cursor,
                    cursor - self->cursor);
```

```
            if (u) {
                token = PyObject_CallMethod(analysis_module, "Token", "OnnO",
                        u,
                        (Py_ssize_t) (self->cursor -
                            PyUnicode_AS_UNICODE(self->input)),
                        (Py_ssize_t) (cursor - PyUnicode_AS_UNICODE(self->input)),
                        type);
                Py_DECREF(u); /* "O" above increfs */
            }
            self->cursor = cursor;
            return token;
        } else {
            return NULL; /* exhausted */
        }
    }

    static PyMethodDef StandardTokenizer_methods[] = {
        {NULL} /* sentinel */
    };

    static PyTypeObject StandardTokenizerType = {
        PyObject_HEAD_INIT(NULL)
        0,                          /* ob_size */
        "lucille.analysis._standard.standard_tokenizer",/* tp_name */
        sizeof(StandardTokenizer),  /* tp_basicsize */
        0,                          /* tp_itemsize */
        (destructor) StandardTokenizer_dealloc, /* tp_dealloc */
        0,                          /* tp_print */
        0,                          /* tp_getattr */
        0,                          /* tp_setattr */
        0,                          /* tp_compare */
        0,                          /* tp_repr */
        0,                          /* tp_as_number */
        0,                          /* tp_as_sequence */
        0,                          /* tp_as_mapping */
        0,                          /* tp_hash  */
        0,                          /* tp_call */
        0,                          /* tp_str */
        0,                          /* tp_getattro */
        0,                          /* tp_setattro */
        0,                          /* tp_as_buffer */
        Py_TPFLAGS_DEFAULT | Py_TPFLAGS_BASETYPE | Py_TPFLAGS_HAVE_ITER, /* tp_flags */
        StandardTokenizer_doc,      /* tp_doc */
        0,                          /* tp_traverse */
```

```
        0,                          /* tp_clear */
211     0,                          /* tp_richcompare */
        0,                          /* tp_weaklistoffset */
        (getiterfunc) StandardTokenizer_iter, /* tp_iter */
        (iternextfunc) StandardTokenizer_iternext, /* tp_iternext */
        StandardTokenizer_methods,  /* tp_methods */
        StandardTokenizer_members,  /* tp_members */
        0,                          /* tp_getset */
        0,                          /* tp_base */
        0,                          /* tp_dict */
        0,                          /* tp_descr_get */
221     0,                          /* tp_descr_set */
        0,                          /* tp_dictoffset */
        (initproc) StandardTokenizer_init, /* tp_init */
        0,                          /* tp_alloc */
        StandardTokenizer_new,      /* tp_new */
    };

    static PyMethodDef _standard_methods[] = {
        {NULL}  /* sentinel */
    };
231
    PyMODINIT_FUNC init_standard(void) {
        PyObject *analysis_module_name = NULL;

        /* create strings */
        if (!(alphanum_type = PyString_FromString("ALPHANUM")))
            goto error;
        if (!(apostrophe_type = PyString_FromString("APOSTROPHE")))
            goto error;
        if (!(acronym_type = PyString_FromString("ACRONYM")))
241         goto error;
        if (!(company_type = PyString_FromString("COMPANY")))
            goto error;
        if (!(email_type = PyString_FromString("EMAIL")))
            goto error;
        if (!(host_type = PyString_FromString("HOST")))
            goto error;
        if (!(num_type = PyString_FromString("NUM")))
            goto error;
        if (!(cj_type = PyString_FromString("CJ")))
251         goto error;

        /* import lucille.analysis */
```

```
    if (!(analysis_module_name = PyString_FromString("lucille.analysis")))
        goto error;
    if (!(analysis_module = PyImport_Import(analysis_module_name)))
        goto error;
    Py_DECREF(analysis_module_name);
    analysis_module_name = NULL;

    /* initialise this module */
    PyObject *m;

    StandardTokenizerType.tp_new = PyType_GenericNew;
    if (PyType_Ready(&StandardTokenizerType) < 0)
        goto error;

    m = Py_InitModule("lucille.analysis._standard", _standard_methods);

    Py_INCREF(&StandardTokenizerType);
    PyModule_AddObject(m, "standard_tokenizer", (PyObject *)&StandardTokenizerType)
        ;

    return;

error:
    Py_XDECREF(alphanum_type);
    Py_XDECREF(apostrophe_type);
    Py_XDECREF(acronym_type);
    Py_XDECREF(company_type);
    Py_XDECREF(email_type);
    Py_XDECREF(host_type);
    Py_XDECREF(num_type);
    Py_XDECREF(cj_type);
    Py_XDECREF(analysis_module);
    Py_XDECREF(analysis_module_name);
    return;
}
```

A.3 CLucene `search` module

`IndexReader.h`: header file for stub `IndexReader` class

```
/*------------------------------------------------------------------------------
* Copyright (C) 2003-2006 Ben van Klinken and the CLucene Team
*
* Distributable under the terms of either the Apache License (Version 2.0) or
* the GNU Lesser General Public License, as specified in the COPYING file.
```

```
------------------------------------------------------------------------------*/
#ifndef _lucene_index_IndexReader_
#define _lucene_index_IndexReader_

#if defined(_LUCENE_PRAGMA_ONCE)
# pragma once
#endif

#include <boost/python.hpp>

//#include "CLucene/store/Directory.h"
//#include "CLucene/store/FSDirectory.h"
//#include "CLucene/store/Lock.h"
//#include "CLucene/document/Document.h"
//#include "CLucene/index/TermVector.h"
//#include "SegmentInfos.h"
#include "Terms.h"


CL_NS_DEF(index)


/** IndexReader is an abstract class, providing an interface for accessing an
 index.  Search of an index is done entirely through this abstract interface,
 so that any subclass which implements it is searchable.

 <p> Concrete subclasses of IndexReader are usually constructed with a call to
 one of the static <code>open()</code> methods, e.g. {@link #open(String)}.

 <p> For efficiency, in this API documents are often referred to via
 <i>document numbers</i>, non-negative integers which each name a unique
 document in the index.  These document numbers are ephemeral--they may change
 as documents are added to and deleted from an index.  Clients should thus not
 rely on a given document having the same number between sessions.

 <p> An IndexReader can be opened on a directory for which an IndexWriter is
 opened already, but it cannot be used to delete documents from the index then.
*/
class IndexReader :LUCENE_BASE{
private:
    boost::python::object _real; // underlying object from Python

public:
```

```
        IndexReader(boost::python::object _real);

        //Callback for classes that need to know if IndexReader is closing.
53      typedef void (*CloseCallback)(IndexReader*, void*);

        class CloseCallbackCompare:public CL_NS(util)::Compare::_base{
        public:
            bool operator()( CloseCallback t1, CloseCallback t2 ) const{
                return t1 > t2;
            }
            static void doDelete(CloseCallback dummy){
            }
        };
63
    #if 0

        enum FieldOption {
            // all fields
            ALL = 1,
            // all indexed fields
            INDEXED = 2,
            // all fields which are not indexed
            UNINDEXED = 4,
73          // all fields which are indexed with termvectors enables
            INDEXED_WITH_TERMVECTOR = 8,
            // all fields which are indexed but don't have termvectors enabled
            INDEXED_NO_TERMVECTOR = 16,
            // all fields where termvectors are enabled. Please note that only standard
                 termvector fields are returned
            TERMVECTOR = 32,
            // all field with termvectors wiht positions enabled
            TERMVECTOR_WITH_POSITION = 64,
            // all fields where termvectors with offset position are set
            TERMVECTOR_WITH_OFFSET = 128,
83          // all fields where termvectors with offset and position values set
            TERMVECTOR_WITH_POSITION_OFFSET = 256
        };

    #endif
    private:
    #if 0
        CL_NS(store)::LuceneLock* writeLock;

        bool directoryOwner;
```

```
93      bool stale;
        bool hasChanges;
        bool closeDirectory;

        SegmentInfos* segmentInfos;

        CL_NS(store)::Directory* directory;
    #endif
        typedef CL_NS(util)::CLSet<CloseCallback, void*,
            CloseCallbackCompare,
103         CloseCallbackCompare> CloseCallbackMap;
        CloseCallbackMap closeCallbacks;

    #if 0
        /** Internal use. Implements commit */
        virtual void doCommit() = 0;

        /**
        * Tries to acquire the WriteLock on this directory.
        * this method is only valid if this IndexReader is directory owner.
113     *
        * @throws IOException If WriteLock cannot be acquired.
        */
        void aquireWriteLock();
    protected:
        /**
        * Constructor used if IndexReader is not owner of its directory.
        * This is used for IndexReaders that are used within other IndexReaders that
            take care or locking directories.
        *
        * @param directory Directory where IndexReader files reside.
123     */
        IndexReader(CL_NS(store)::Directory* dir);

        /**
        * Constructor used if IndexReader is owner of its directory.
        * If IndexReader is owner of its directory, it locks its directory in case of
            write operations.
        *
        * @param directory Directory where IndexReader files reside.
        * @param segmentInfos Used for write-l
        * @param closeDirectory
133     */
        IndexReader(CL_NS(store)::Directory* directory, SegmentInfos* segmentInfos,
```

```
        bool closeDirectory);


    /// Implements close.
    virtual void doClose() = 0;

    /** Implements setNorm in subclass.*/
    virtual void doSetNorm(int32_t doc, const TCHAR* field, uint8_t value) = 0;

    /** Implements actual undeleteAll() in subclass. */
    virtual void doUndeleteAll() = 0;


    /** Implements deletion of the document numbered <code>docNum</code>.
    * Applications should call {@link #deleteDocument(int32_t)} or {@link #
        deleteDocuments(Term*)}.
    */
    virtual void doDelete(const int32_t docNum) = 0;

#endif
public:

#if 0
    DEFINE_MUTEX(THIS_LOCK)

    ///Do not access this directly, only public so that MultiReader can access it
    virtual void commit();


    /** Undeletes all documents currently marked as deleted in this index.*/
    void undeleteAll();

    /**
    * Get a list of unique field names that exist in this index and have the
        specified
    * field option information.
    * @param fldOption specifies which field option should be available for the
        returned fields
    * @return Collection of Strings indicating the names of the fields.
    * @see IndexReader.FieldOption
    */
    virtual void getFieldNames(FieldOption fldOption, CL_NS(util)::
        StringArrayWithDeletor& retarray) = 0;

```

```
    _CL_DEPRECATED( getFieldNames(FieldOption, StringArrayWithDeletor&) ) virtual
        TCHAR** getFieldNames();
    _CL_DEPRECATED( getFieldNames(FieldOption, StringArrayWithDeletor&) ) virtual
        TCHAR** getFieldNames(bool indexed);
#endif

    /** Returns the byte-encoded normalization factor for the named field of
    * every document.  This is used by the search code to score documents.
    *
    * The number of bytes returned is the size of the IndexReader->maxDoc()
    * MEMORY: The values are cached, so don't delete the returned byte array.
183 * @see Field#setBoost(float_t)
    */
    const uint8_t* norms(const TCHAR* field);

#if 0
    /** Reads the byte-encoded normalization factor for the named field of every
    *  document.  This is used by the search code to score documents.
    *
    * @see Field#setBoost(float_t)
    */
193 virtual void norms(const TCHAR* field, uint8_t* bytes) = 0;

    /** Expert: Resets the normalization factor for the named field of the named
    * document.
    *
    * @see #norms(TCHAR*)
    * @see Similarity#decodeNorm(uint8_t)
    */
    void setNorm(int32_t doc, const TCHAR* field, float_t value);

203 /** Expert: Resets the normalization factor for the named field of the named
    * document.  The norm represents the product of the field's {@link
    * Field#setBoost(float_t) boost} and its {@link Similarity#lengthNorm(TCHAR*,
    * int32_t) length normalization}.  Thus, to preserve the length normalization
    * values when resetting this, one should base the new value upon the old.
    *
    * @see #norms(TCHAR*)
    * @see Similarity#decodeNorm(uint8_t)
    */
    void setNorm(int32_t doc, const TCHAR* field, uint8_t value);
213
    /// Release the write lock, if needed.
    virtual ~IndexReader();
```

```
        /// Returns an IndexReader reading the index in an FSDirectory in the named
            path.
        static IndexReader* open(const char* path);

        /// Returns an IndexReader reading the index in the given Directory.
        static IndexReader* open( CL_NS(store)::Directory* directory, bool
            closeDirectory=false);

223     /**
        * Returns the time the index in the named directory was last modified.
        * Do not use this to check whether the reader is still up-to-date, use
        * {@link #isCurrent()} instead.
        */
        static uint64_t lastModified(const char* directory);

        /**
        * Returns the time the index in the named directory was last modified.
        * Do not use this to check whether the reader is still up-to-date, use
233     * {@link #isCurrent()} instead.
        */
        static uint64_t lastModified(const CL_NS(store)::Directory* directory);


        /**
        * Reads version number from segments files. The version number is
        * initialized with a timestamp and then increased by one for each change of
        * the index.
        *
243     * @param directory where the index resides.
        * @return version number.
        * @throws IOException if segments file cannot be read
        */
        static int64_t getCurrentVersion(CL_NS(store)::Directory* directory);

        /**
       * Reads version number from segments files. The version number is
       * initialized with a timestamp and then increased by one for each change of
       * the index.
253    *
       * @param directory where the index resides.
       * @return version number.
       * @throws IOException if segments file cannot be read
       */
```

```
        static int64_t getCurrentVersion(const char* directory);

        /**
        * Version number when this IndexReader was opened.
        */
263     int64_t getVersion();

        /**
        * Check whether this IndexReader still works on a current version of the index.
        * If this is not the case you will need to re-open the IndexReader to
        * make sure you see the latest changes made to the index.
        *
        * @throws IOException
        */
        bool isCurrent();
273

        /**
        *  Return an array of term frequency vectors for the specified document.
        *  The array contains a vector for each vectorized field in the document.
        *  Each vector contains terms and frequencies for all terms in a given
            vectorized field.
        *  If no such fields existed, the method returns null. The term vectors that
            are
        * returned my either be of type TermFreqVector or of type TermPositionsVector
            if
        * positions or offsets have been stored.
        *
283     * @param docNumber document for which term frequency vectors are returned
        * @return array of term frequency vectors. May be null if no term vectors have
            been
        *  stored for the specified document.
        * @throws IOException if index cannot be accessed
        * @see org.apache.lucene.document.Field.TermVector
        */
        virtual bool getTermFreqVectors(int32_t docNumber, Array<TermFreqVector*>&
            result) =0;

        /**
        *  Return a term frequency vector for the specified document and field. The
293     *  returned vector contains terms and frequencies for the terms in
        *  the specified field of this document, if the field had the storeTermVector
        *  flag set. If termvectors had been stored with positions or offsets, a
        *  TermPositionsVector is returned.
```

```
        *
        * @param docNumber document for which the term frequency vector is returned
        * @param field field for which the term frequency vector is returned.
        * @return term frequency vector May be null if field does not exist in the
            specified
        * document or term vector was not stored.
        * @throws IOException if index cannot be accessed
303     * @see org.apache.lucene.document.Field.TermVector
        */
        virtual TermFreqVector* getTermFreqVector(int32_t docNumber, const TCHAR* field
            ) = 0;

        /**
        * Returns <code>true</code> if an index exists at the specified directory.
        * If the directory does not exist or if there is no index in it.
        * @param  directory the directory to check for an index
        * @return <code>true</code> if an index exists; <code>false</code> otherwise
        */
313     static bool indexExists(const char* directory);

        /**
        * Returns <code>true</code> if an index exists at the specified directory.
        * If the directory does not exist or if there is no index in it.
        * @param  directory the directory to check for an index
        * @return <code>true</code> if an index exists; <code>false</code> otherwise
        * @throws IOException if there is a problem with accessing the index
        */
        static bool indexExists(const CL_NS(store)::Directory* directory);
323
        /** Returns the number of documents in this index. */
        virtual int32_t numDocs() = 0;
    #endif

        /** Returns one greater than the largest possible document number.
        * This may be used to, e.g., determine how big to allocate an array which
        * will have an element for every document number in an index.
        */
        int32_t maxDoc() const;
333
        boost::python::object document(int32_t n);

    #if 0
        /** Gets the stored fields of the <code>n</code><sup>th</sup>
         * <code>Document</code> in this index.
```

```
         * The fields are not cleared before retrieving the document, so the
         * object should be new or just cleared.
         */
        virtual bool document(int32_t n, CL_NS(document)::Document*) =0;
343
        _CL_DEPRECATED( document(i, document) ) CL_NS(document)::Document* document(
            const int32_t n);

        /** Returns true if document <i>n</i> has been deleted */
        virtual bool isDeleted(const int32_t n) = 0;

        /** Returns true if any documents have been deleted */
        virtual bool hasDeletions() const = 0;

        /** Returns true if there are norms stored for this field. */
353     virtual bool hasNorms(const TCHAR* field);
    #endif

        /** Returns an enumeration of all the terms in the index.
        * The enumeration is ordered by Term.compareTo().  Each term
        * is greater than all that precede it in the enumeration.
        * @memory Caller must clean up
        */
        TermEnum* terms() const;

363     /** Returns an enumeration of all terms after a given term.
        * The enumeration is ordered by Term.compareTo().  Each term
        * is greater than all that precede it in the enumeration.
        * @memory Caller must clean up
        */
        TermEnum* terms(const Term* t) const;

        /** Returns the number of documents containing the term <code>t</code>. */
        int32_t docFreq(const Term* t) const;

373 #if 0
        /* Returns an unpositioned TermPositions enumerator.
         * @memory Caller must clean up
         */
        virtual TermPositions* termPositions() const = 0;

        /** Returns an enumeration of all the documents which contain
        * <code>term</code>.  For each document, in addition to the document number
        * and frequency of the term in that document, a list of all of the ordinal
```

```
        * positions of the term in the document is available.  Thus, this method
383     * implements the mapping:
        *
        * <p><ul>
        * Term    =&gt;    &lt;docNum, freq,
        * &lt;pos<sub>1</sub>, pos<sub>2</sub>, ...
        * pos<sub>freq-1</sub>&gt;
        * &gt;<sup>*</sup>
        * </ul>
        * <p> This positional information faciliates phrase and proximity searching.
        * <p>The enumeration is ordered by document number.  Each document number is
393     * greater than all that precede it in the enumeration.
         * @memory Caller must clean up
        */
        TermPositions* termPositions(Term* term) const;
    #endif

        /** Returns an unpositioned {@link TermDocs} enumerator.
         * @memory Caller must clean up
         */
        TermDocs* termDocs() const;
403
        /** Returns an enumeration of all the documents which contain
        * <code>term</code>. For each document, the document number, the frequency of
        * the term in that document is also provided, for use in search scoring.
        * Thus, this method implements the mapping:
        * <p><ul>Term    =&gt;    &lt;docNum, freq&gt;<sup>*</sup
            ></ul>
        * <p>The enumeration is ordered by document number.  Each document number
        * is greater than all that precede it in the enumeration.
         * @memory Caller must clean up
        */
413     TermDocs* termDocs(const Term* term) const;

    #if 0
        /** Deletes the document numbered <code>docNum</code>.  Once a document is
        * deleted it will not appear in TermDocs or TermPostitions enumerations.
        * Attempts to read its field with the {@link #document}
        * method will result in an error.  The presence of this document may still be
        * reflected in the {@link #docFreq} statistic, though
        * this will be corrected eventually as the index is further modified.
        */
423     void deleteDocument(const int32_t docNum);
```

```
        ///@deprecated. Use deleteDocument instead.
        _CL_DEPRECATED( deleteDocument ) void deleteDoc(const int32_t docNum){
            deleteDocument(docNum); }

        /** Deletes all documents containing <code>term</code>.
        * This is useful if one uses a document field to hold a unique ID string for
        * the document.  Then to delete such a document, one merely constructs a
        * term with the appropriate field and the unique ID string as its text and
        * passes it to this method.
433     * See {@link #deleteDocument(int)} for information about when this deletion
            will
        * become effective.
        * @return the number of documents deleted
        */
        int32_t deleteDocuments(Term* term);

        ///@deprecated. Use deleteDocuments instead.
        _CL_DEPRECATED( deleteDocuments ) int32_t deleteTerm(Term* term){ return
            deleteDocuments(term); }
    #endif

443     /**
        * Closes files associated with this index and also saves any new deletions to
            disk.
        * No other methods should be called after this has been called.
        */
        void close();

    #if 0
        ///Checks if the index in the named directory is currently locked.
        static bool isLocked(CL_NS(store)::Directory* directory);

453     ///Checks if the index in the named directory is currently locked.
        static bool isLocked(const char* directory);


        ///Forcibly unlocks the index in the named directory.
        ///Caution: this should only be used by failure recovery code,
        ///when it is known that no other process nor thread is in fact
        ///currently accessing this index.
        static void unlock(CL_NS(store)::Directory* directory);
        static void unlock(const char* path);
463
         /** Returns the directory this index resides in. */
```

```
        CL_NS(store)::Directory* getDirectory() { return directory; }

        /** Returns true if the file is a lucene filename (based on extension or
            filename) */
        static bool isLuceneFile(const char* filename);
    #endif

        /**
        * For classes that need to know when the IndexReader closes (such as caches,
            etc),
473     * should pass their callback function to this.
        */
        void addCloseCallback(CloseCallback callback, void* parameter);

    #if 0
    protected:
        class LockWith:public CL_NS(store)::LuceneLockWith<IndexReader*>{
        public:
            CL_NS(store)::Directory* directory;
            IndexReader* indexReader;
483
            //Constructor
            LockWith(CL_NS(store)::LuceneLock* lock, CL_NS(store)::Directory* dir);

            //Reads the segmentinfo file and depending on the number of segments found
            //it returns a MultiReader or a SegmentReader
            IndexReader* doBody();

        };
        friend class IndexReader::LockWith;
493
        class CommitLockWith:public CL_NS(store)::LuceneLockWith<void>{
        private:
            IndexReader* reader;
        public:

            //Constructor
            CommitLockWith( CL_NS(store)::LuceneLock* lock, IndexReader* r );
            void doBody();
        };
503     friend class IndexReader::CommitLockWith;
    #endif
    };
```

```
CL_NS_END
#endif
```

IndexReader.cpp: C++ source file implementing stub IndexReader class

```
#include "CLucene/StdHeader.h"
#include "IndexReader.h"

CL_NS_DEF(index)

  IndexReader::IndexReader(boost::python::object _real)
    : _real(_real)
  {
  }

  const uint8_t *IndexReader::norms(const TCHAR *field) {
    return (const uint8_t *)PyString_AsString(_real.attr("norms")(wstring(field)).
        ptr());
  }

  int32_t IndexReader::maxDoc() const {
    return boost::python::extract<int32_t>(_real.attr("max_doc")());
  }

  TermEnum *IndexReader::terms() const {
    return _CLNEW TermEnum(_real.attr("terms")());
  }

  TermEnum* IndexReader::terms(const Term *t) const {
    TermEnum *te = terms();
    te->skipTo(t);
    return te;
  }

  boost::python::object IndexReader::document(int32_t n) {
    return _real.attr("document")(n);
  }

  int32_t IndexReader::docFreq(const Term *t) const {
    return boost::python::extract<int32_t>(_real.attr("doc_freq")(t->asPythonTerm()
        ));
  }

  TermDocs *IndexReader::termDocs() const {
    return _CLNEW TermDocs(_real.attr("term_docs")());
```

```
  }

  TermDocs *IndexReader::termDocs(const Term *t) const {
    TermDocs *td = termDocs();
    td->seek(t);
    return td;
  }

  void IndexReader::close() {
    CloseCallbackMap::iterator iter = closeCallbacks.begin();
    for ( ;iter!=closeCallbacks.end();iter++){
        CloseCallback callback = *iter->first;
        callback(this,iter->second);
    }
  }

    void IndexReader::addCloseCallback(CloseCallback callback, void* parameter){
        closeCallbacks.put(callback, parameter);
    }

CL_NS_END
```

`Term.h`: header file for `Term` wrapper class

```
/*------------------------------------------------------------------------------
* Copyright (C) 2003-2006 Ben van Klinken and the CLucene Team
*
* Distributable under the terms of either the Apache License (Version 2.0) or
* the GNU Lesser General Public License, as specified in the COPYING file.
------------------------------------------------------------------------------*/
#ifndef _lucene_index_Term_
#define _lucene_index_Term_

#if defined(_LUCENE_PRAGMA_ONCE)
# pragma once
#endif

#include <boost/python.hpp>

#include "CLucene/util/Misc.h"
#include "CLucene/util/StringIntern.h"

CL_NS_DEF(index)

/**
```

```
    A Term represents a word from text.  This is the unit of search.  It is
    composed of two elements, the text of the word, as a string, and the name of
    the field that the text occured in, an interned string.

    Note that terms may represent more than words from text fields, but also
    things like dates, email addresses, urls, etc.

    IMPORTANT NOTE:
    Term inherits from the template class LUCENE_REFBASE which tries to do
31  some garbage collection by counting the references an instance has. As a result
    of this construction you MUST use _CLDECDELETE(obj) when you want to delete an
    of Term!

    ABOUT intrn

    intrn indicates if field and text are interned or not. Interning of Strings is the
        process of
    converting duplicated strings to shared ones.

    */
41  class Term:LUCENE_REFBASE {
      private:
        size_t cachedHashCode;
        const TCHAR* _field;
        //CLStringIntern::iterator fielditr;
    #ifdef LUCENE_TERM_TEXT_LENGTH
        TCHAR _text[LUCENE_TERM_TEXT_LENGTH+1];
    #else
        TCHAR* _text;
        size_t textLenBuf; //a cache of text len, this allows for a preliminary
            comparison of text lengths
51      //bool    dupT;    //Indicates if Term Text is duplicated (and therefore must
            be deleted).
    #endif
        size_t textLen; //a cache of text len, this allows for a preliminary comparison
             of text lengths
        bool    internF; //Indicates if Term Field is interned(and therefore must be
            uninternd).
      public:

        //uses the specified fieldTerm's field. this saves on intern'ing time.
        Term(const Term* fieldTerm, const TCHAR* txt);

        ///Constructs a blank term
```

```
61      Term();

        //todo: these need to be private, but a few other things need to be changed
            first...
        Term(const TCHAR* fld, const TCHAR* txt, bool internField);

        /**
        * Constructor. Constructs a Term with the given field and text. Field and text
            are not copied
        * Field and text are deleted in destructor only if intern is false.
        */
        Term(const TCHAR* fld, const TCHAR* txt);
71
        ///Destructor.
        ~Term();

        ///Returns the field of this term, an interned string. The field indicates
        ///the part of a document which this term came from.
        const TCHAR* field() const; ///<returns reference

        ///Returns the text of this term.  In the case of words, this is simply the
        ///text of the word.  In the case of dates and other types, this is an
81      ///encoding of the object as a string.
        const TCHAR* text() const; ///<returns reference

        boost::python::object _fieldPython() const;
        boost::python::object _textPython() const;

        ///Resets the field and text of a Term.
        inline void set(const TCHAR* fld, const TCHAR* txt){
            set(fld,txt,true);
        }
91
        /**
        * Optimized set of Term by reusing same field as this Term
        * - avoids field.intern() overhead
        * @param text The text of the new term (field is implicitly same as this Term
            instance)
        */
        void set(const Term* term, const TCHAR* txt);

        void set(const TCHAR* fld, const TCHAR* txt, const bool internField);

101     void set(boost::python::object _term);
```

```
        boost::python::tuple asPythonTerm() const;

        /** Compares two terms, returning a negative integer if this
        term belongs before the argument, zero if this term is equal to the
        argument, and a positive integer if this term belongs after the argument.

        The ordering of terms is first by field, then by text.*/
        int32_t compareTo(const Term* other) const;
111
        bool equals(const Term* other) const;

        size_t textLength() const { return textLen; }

        ///Forms the contents of Field and term in some kind of tuple notation
        ///<field:text>
        TCHAR* toString() const;

        size_t hashCode();
121



        class Equals:public CL_NS_STD(binary_function)<const Term*,const Term*,bool>
        {
        public:
            bool operator()( const Term* val1, const Term* val2 ) const{
                return val1->equals(val2);
            }
        };
131
        class Compare:LUCENE_BASE, public CL_NS(util)::Compare::_base //<Term*>
        {
        public:
            bool operator()( Term* t1, Term* t2 ) const{
                return ( t1->compareTo(t2) < 0 );
            }
            size_t operator()( Term* t ) const{
                return t->hashCode();
            }
141     };
    };

    CL_NS_END
    #endif
```

Term.cpp: C++ source file implementing Term wrapper class

```
/*------------------------------------------------------------------------------
* Copyright (C) 2003-2006 Ben van Klinken and the CLucene Team
*
* Distributable under the terms of either the Apache License (Version 2.0) or
* the GNU Lesser General Public License, as specified in the COPYING file.
------------------------------------------------------------------------------*/
#include "CLucene/StdHeader.h"
#include "Term.h"
#include "CLucene/util/StringIntern.h"

CL_NS_USE(util)
CL_NS_DEF(index)

Term::Term(){
    //Intern fld and assign it to field
    _field = LUCENE_BLANK_STRING;
    internF = false;
    cachedHashCode = 0;
    textLen = 0;

    //Duplicate txt and assign it to text
    #ifdef LUCENE_TERM_TEXT_LENGTH
        _text[0]=0;
    #else
        _text = LUCENE_BLANK_STRING;
        textLenBuf = 0;
    #endif
    textLen = 0;
}

Term::Term(const TCHAR* fld, const TCHAR* txt,const bool internField){
//Func - Constructor.
//       Constructs a Term with the given field and text. Field and text are not
    copied
//       Field and text are deleted in destructor only if intern is false.
//Pre  - fld != NULL and contains the name of the field
//       txt != NULL and contains the value of the field
//       internF is true or false and indicates if term Field is interned or not
//       internT is true or false and indicates if term Text is interned or not
//       canDelete defaults to true but can be false and indicates to the
    IGarbageCollector that the Term can be deleted when finalized
//Post - An instance of Term has been created.Field and txt have not been copied
    but assigned
```

```
        _field = LUCENE_BLANK_STRING;
        internF = false;
        textLen = 0;
        #ifdef LUCENE_TERM_TEXT_LENGTH
            _text[0]=0;
        #else
            _text = LUCENE_BLANK_STRING;
            textLenBuf = 0;
        #endif

        set(fld,txt,internField);
    }

    Term::Term(const Term* fieldTerm, const TCHAR* txt){
        _field = LUCENE_BLANK_STRING;
        internF = false;
        textLen = 0;
        #ifdef LUCENE_TERM_TEXT_LENGTH
            _text[0]=0;
        #else
            _text = LUCENE_BLANK_STRING;
            textLenBuf = 0;
        #endif

        set(fieldTerm,txt);
    }

    Term::Term(const TCHAR* fld, const TCHAR* txt){
        _field = LUCENE_BLANK_STRING;
        internF = false;
        textLen = 0;
        #ifdef LUCENE_TERM_TEXT_LENGTH
            _text[0]=0;
        #else
            _text = LUCENE_BLANK_STRING;
            textLenBuf = 0;
        #endif

        set(fld,txt);
    }

    Term::~Term(){
```

```
//Func - Destructor.
//Pre  - true
//Post - The instance has been destroyed. field and text have been deleted if pre(
    intrn) is false

    //Unintern field
    if ( internF )
        CLStringIntern::unintern(_field);
    _field = NULL;

#ifndef LUCENE_TERM_TEXT_LENGTH
    //Deletetext if it is the owner
    if ( _text != LUCENE_BLANK_STRING)
        _CLDELETE_CARRAY( _text );
#endif
}

const TCHAR* Term::field() const {
//Func - Returns the field of this term, an interned string. The field indicates
//       the part of a document which this term came from.
//Pre  - true
//Post - field has been returned

    return _field;
}

const TCHAR* Term::text() const {
//Func - Returns the text of this term.  In the case of words, this is simply the
//       text of the word.  In the case of dates and other types, this is an
//       encoding of the object as a string.
//Pre  - true
//Post - text has been returned

    return _text;
}

boost::python::object Term::_fieldPython() const {
    return boost::python::object(wstring(field()));
}

boost::python::object Term::_textPython() const {
    return boost::python::object(wstring(text()));
}
```

```
void Term::set(boost::python::object _term) {
    const wstring field = boost::python::extract<wstring>(_term[0]);
    const wstring text = boost::python::extract<wstring>(_term[1]);
    set(field.c_str(), text.c_str());
}

boost::python::tuple Term::asPythonTerm() const {
    return boost::python::make_tuple(wstring(_field), wstring(_text));
}

void Term::set(const Term* term, const TCHAR* txt){
    set(term->field(),txt,false);
}

void Term::set(const TCHAR* fld, const TCHAR* txt,const bool internField){
//Func - Resets the field and text of a Term.
//Pre  - fld != NULL and contains the name of the field
//       txt != NULL and contains the value of the field
//       internF is true or false
//       internT is true or false
//Post - field and text of Term have been reset

    CND_PRECONDITION(fld != NULL, "fld contains NULL");
    CND_PRECONDITION(txt != NULL, "txt contains NULL");

    //save field for unintern later
    const TCHAR* oldField = _field;
    //bool oldInternF = internF; //Not used
    cachedHashCode = 0;

    textLen = _tcslen(txt);

    //Delete text if it is the owner
#ifdef LUCENE_TERM_TEXT_LENGTH
    if ( textLen > LUCENE_TERM_TEXT_LENGTH )
       textLen = LUCENE_TERM_TEXT_LENGTH;
    _tcsncpy(_text,txt,textLen+1);
    _text[textLen]=0;
#else

    //if the term text buffer is bigger than what we have
    if ( _text && textLen > textLenBuf){
        if ( _text != LUCENE_BLANK_STRING ){
            _CLDELETE_ARRAY( _text );
```

```
            }else
                _text = NULL;
            textLenBuf = 0;
        }

        if ( _text==LUCENE_BLANK_STRING )
            _text = LUCENE_BLANK_STRING;
        else if ( _text==NULL ){
            if ( txt[0] == 0 ){
                //if the string is blank and we aren't re-using the buffer...
                _text = LUCENE_BLANK_STRING;
            }else{
                //duplicate the text
                _text  = stringDuplicate(txt);
                textLenBuf = textLen;
            }
        }else{
            //re-use the buffer
            _tcscpy(_text,txt);
        }

    #endif

        //Set Term Field
        if ( internField )
            _field = CLStringIntern::intern(fld  CL_FILELINE);
        else
            _field = fld;

        //unintern old field after interning new one,
        if ( internF )
            CLStringIntern::unintern(oldField);
        internF = internField;

        CND_PRECONDITION(_tcscmp(fld, _field)==0,"field not equal");
    }

    /** Compares two terms, returning true iff they have the same
      field and text. */
    bool Term::equals(const Term* other) const{
       if ( cachedHashCode != 0 && other->cachedHashCode != 0 &&
            other->cachedHashCode != cachedHashCode )
            return false;
```

```
  if ( _field==other->_field ){
     //this can be quicker than using compareTo, because checks
     //field length first
      if ( textLen == other->textLen ){
          return (_tcscmp(_text,other->_text)==0);
      }else
        return false;
  }else
     return false;
}

size_t Term::hashCode(){
    if ( cachedHashCode == 0 )
        cachedHashCode = Misc::thashCode(_field) + Misc::thashCode(_text,textLen);
    return cachedHashCode;
}


int32_t Term::compareTo(const Term* other) const {
//Func - Compares two terms, to see if this term belongs before,is equal to or
    after
//       after the argument term.
//Pre  - other is a reference to another term
//Post - A negative integer is returned if this term belongs before the argument,
//       zero is returned if this term is equal to the argument, and a positive
    integer
//       if this term belongs after the argument.

    //Check ret to see if text needs to be compared
    if ( _field == other->_field ){ // fields are interned
        //Compare text with text of other and return the result
        return _tcscmp(_text,other->_text);
    }else
        return _tcscmp(_field,other->_field);
}

TCHAR* Term::toString() const{
//Func - Forms the contents of Field and term in some kind of tuple notation
//       <field:text>
//Pre  - true
//Post - a string formatted as <field:text> is returned if pre(field) is NULL and
//       text is NULL the returned string will be formatted as <:>

    return CL_NS(util)::Misc::join( _field, _T(":"), _text);
```

```
}

CL_NS_END
```

`Terms.h`: header file for stub `TermDocs` and `TermEnum` classes

```
/*------------------------------------------------------------------------------
* Copyright (C) 2003-2006 Ben van Klinken and the CLucene Team
*
* Distributable under the terms of either the Apache License (Version 2.0) or
* the GNU Lesser General Public License, as specified in the COPYING file.
------------------------------------------------------------------------------*/
#ifndef _lucene_index_Terms_
#define _lucene_index_Terms_

#if defined(_LUCENE_PRAGMA_ONCE)
# pragma once
#endif

#include <boost/python.hpp>

#include "Term.h"
CL_NS_DEF(index)

class TermEnum; //predefine
class TermPositions;

/** TermDocs provides an interface for enumerating &lt;document, frequency&gt;
 pairs for a term.  <p> The document portion names each document containing
 the term.  Documents are indicated by number.  The frequency portion gives
 the number of times the term occurred in each document.  <p> The pairs are
 ordered by document number.

 @see IndexReader#termDocs()
 */
class TermDocs: LUCENE_BASE {
private:
    boost::python::object _real; // underlying object from Python

public:
#if 0
    virtual ~TermDocs(){
    }
#endif
```

```
40      // Sets this to the data for a term.
        // The enumeration is reset to the start of the data for this term.
        void seek(const Term* term);

        TermDocs(boost::python::object _real);

        /** Sets this to the data for the current term in a {@link TermEnum}.
        * This may be optimized in some implementations.
        */
        void seek(const TermEnum* termEnum);
50
        // Returns the current document number.  <p> This is invalid until {@link
        //  #next()} is called for the first time.
        int32_t doc() const;

        // Returns the frequency of the term within the current document.  <p> This
        //  is invalid until {@link #next()} is called for the first time.
        int32_t freq() const;

        // Moves to the next pair in the enumeration.  <p> Returns true iff there is
60      //  such a next pair in the enumeration.
        bool next();

        // Attempts to read multiple entries from the enumeration, up to length of
        // <i>docs</i>.  Document numbers are stored in <i>docs</i>, and term
        // frequencies are stored in <i>freqs</i>.  The <i>freqs</i> array must be as
        // int64_t as the <i>docs</i> array.
        //
        // <p>Returns the number of entries read.  Zero is only returned when the
        // stream has been exhausted.
70      int32_t read(int32_t* docs, int32_t* freqs, int32_t length);

        // Skips entries to the first beyond the current whose document number is
        // greater than or equal to <i>target</i>. <p>Returns true iff there is such
        // an entry.  <p>Behaves as if written: <pre>
        //   bool skipTo(int32_t target) {
        //     do {
        //       if (!next())
        //       return false;
        //     } while (target > doc());
80      //     return true;
        //   }
        // </pre>
        // Some implementations are considerably more efficient than that.
```

```
        bool skipTo(const int32_t target);

        // Frees associated resources.
        void close();


90  #if 0
        /** Solve the diamond inheritence problem by providing a reinterpret function.
        *   No dynamic casting is required and no RTTI data is needed to do this
        */
        virtual TermPositions* __asTermPositions()=0;
    #endif
    };

    // Abstract class for enumerating terms.
    //
100 //<p>Term enumerations are always ordered by Term.compareTo().  Each term in
    //the enumeration is greater than all that precede it.
    class TermEnum: LUCENE_BASE {
    private:
        boost::python::object _real; // underlying object from Python
        Term *_term;

    public:

        TermEnum(boost::python::object _real);

110
        // Increments the enumeration to the next element.  True if one exists.
        bool next();

        // Returns a pointer to the current Term in the enumeration.
        Term* term();

        // Returns the current Term in the enumeration.
        Term* term(bool pointer);

120     boost::python::object pythonTerm() const;

    #if 0
        // Returns the docFreq of the current Term in the enumeration.
        virtual int32_t docFreq() const=0;
    #endif

        // Closes the enumeration to further activity, freeing resources.
```

```
        void close();

130 #if 0
        virtual ~TermEnum(){
        }
    #endif

        // Term Vector support
        /** Skips terms to the first beyond the current whose value is
        * greater or equal to <i>target</i>. <p>Returns true iff there is such
        * an entry.  <p>Behaves as if written: <pre>
        *   public boolean skipTo(Term target) {
140     *     do {
        *       if (!next())
        *         return false;
        *     } while (target > term());
        *     return true;
        *   }
        * </pre>
        * Some implementations are considerably more efficient than that.
        */
        void skipTo(const Term* target);
150
    #if 0
        /**
        * Because we need to know how to cast the object, we need the objects name.
        */
        virtual const char* getObjectName() = 0;
    #endif
    };



160
    #if 0
    /**
     * TermPositions provides an interface for enumerating the &lt;document,
     * frequency, &lt;position&gt;* &gt; tuples for a term.  <p> The document and
     * frequency are the same as for a TermDocs.  The positions portion lists the
         ordinal
     * positions of each occurrence of a term in a document.
     *
     * @see IndexReader#termPositions()
     */
170 class TermPositions: public virtual TermDocs {
```

```
    public:
        // Returns next position in the current document.  It is an error to call
        //  this more than {@link #freq()} times
        //  without calling {@link #next()}<p> This is
        //  invalid until {@link #next()} is called for
        //  the first time.
        virtual int32_t nextPosition() = 0;

        virtual ~TermPositions(){
    }

        /** Solve the diamond inheritence problem by providing a reinterpret function.
          * No dynamic casting is required and no RTTI data is needed to do this
          */
        virtual TermDocs* __asTermDocs()=0;
        virtual TermPositions* __asTermPositions()=0;
    };
    #endif
    CL_NS_END
    #endif
```

`TermDocs.cpp`: C++ source file implementing stub `TermDocs` class

```
    #include "CLucene/StdHeader.h"
    #include "Terms.h"

    CL_NS_DEF(index)

    TermDocs::TermDocs(boost::python::object _real)
        : _real(_real)
    {
    }

    void TermDocs::seek(const TermEnum *termEnum) {
        _real.attr("seek")(termEnum->pythonTerm());
    }

    void TermDocs::seek(const Term *term) {
        _real.attr("seek")(term->asPythonTerm());
    }

    int32_t TermDocs::doc() const {
        return boost::python::extract<int32_t>(_real.attr("doc"));
    }
```

```
int32_t TermDocs::freq() const {
    return boost::python::extract<int32_t>(_real.attr("freq"));
}

bool TermDocs::next() {
    return boost::python::extract<bool>(_real.attr("advance")());
}

int32_t TermDocs::read(int32_t* docs, int32_t* freqs, int32_t length) {
    int32_t i = 0;
    while (boost::python::extract<bool>(_real.attr("advance")())) {
        docs[i] = boost::python::extract<int32_t>(_real.attr("doc"));
        freqs[i] = boost::python::extract<int32_t>(_real.attr("freq"));
        i ++;
        if (i >= length) break;
    }
    return i;
}

bool TermDocs::skipTo(const int32_t target) {
    return boost::python::extract<bool>(_real.attr("skip_to")(target));
}

void TermDocs::close() {
}

CL_NS_END
```

TermEnum.cpp: C++ source file implementing stub TermEnum class

```
#include "CLucene/StdHeader.h"
#include "Terms.h"
#include "Term.h"

CL_NS_DEF(index)

TermEnum::TermEnum(boost::python::object _real)
    : _real(_real), _term(_CLNEW Term)
{
}

bool TermEnum::next() {
    if (boost::python::extract<bool>(_real.attr("advance")())) {
        _term->set(_real.attr("term"));
        return true;
```

```
        } else
            return false;
    }

    Term *TermEnum::term() {
21      return _CL_POINTER(_term);
    }

    Term *TermEnum::term(bool pointer) {
        if (pointer)
            return _CL_POINTER(_term);
        else
            return _term;
    }

31  boost::python::object TermEnum::pythonTerm() const {
        return _real.attr("term");
    }

    void TermEnum::close() {
    }

    void TermEnum::skipTo(const Term *t) {
        _real.attr("scan_to")(t->asPythonTerm());
    }
41
    CL_NS_END
```

Patch of changes to `search` module to accomodate stub classes

```
Index: MultiSearcher.cpp
===================================================================
--- MultiSearcher.cpp   (.../vendor/clucene-core-0.9.20/src/CLucene/search/
    MultiSearcher.cpp)   (revision 115)
+++ MultiSearcher.cpp   (.../branches/clucene-search/src/lucille/search/CLucene/
    search/MultiSearcher.cpp)   (revision 115)
@@ -9,13 +9,13 @@

 #include "SearchHeader.h"
8  #include "HitQueue.h"
-#include "CLucene/document/Document.h"
+//#include "CLucene/document/Document.h"
 #include "CLucene/index/Term.h"
 #include "FieldDocSortedHitQueue.h"
```

```
 CL_NS_USE(index)
 CL_NS_USE(util)
-CL_NS_USE(document)
+//CL_NS_USE(document)
18
 CL_NS_DEF(search)

@@ -57,12 +57,19 @@
     return docFreq;
   }

+#if 0
   /** For use by {@link HitCollector} implementations. */
   bool MultiSearcher::doc(int32_t n, Document* d) {
28   int32_t i = subSearcher(n);              // find searcher index
     return searchables[i]->doc(n - starts[i], d);    // dispatch to searcher
   }
+#endif

+  boost::python::object MultiSearcher::doc(const int32_t n) {
+    int32_t i = subSearcher(n);              // find searcher index
+    return searchables[i]->doc(n - starts[i]);   // dispatch to searcher
+  }
+
38   int32_t MultiSearcher::searcherIndex(int32_t n) const{
     return subSearcher(n);
   }
Index: Hits.cpp
===================================================================
--- Hits.cpp    (.../vendor/clucene-core-0.9.20/src/CLucene/search/Hits.cpp)    (
    revision 115)
+++ Hits.cpp    (.../branches/clucene-search/src/lucille/search/CLucene/search/Hits
    .cpp)    (revision 115)
@@ -7,12 +7,12 @@
 #include "CLucene/StdHeader.h"

48 #include "SearchHeader.h"
-#include "CLucene/document/Document.h"
+//#include "CLucene/document/Document.h"
 #include "CLucene/index/IndexReader.h"
 #include "Filter.h"
 #include "CLucene/search/SearchHeader.h"

-CL_NS_USE(document)
```

```
+//CL_NS_USE(document)
 CL_NS_USE(util)
 CL_NS_USE(index)

@@ -26,7 +26,7 @@

         next  = NULL;
         prev  = NULL;
-        doc   = NULL;
+        doc   = boost::python::object();
         score = s;
         id    = i;
     }
@@ -36,7 +36,6 @@
     //Pre  - true
     //Post - The instance has been destroyed

-        _CLDELETE(doc);
     }


@@ -66,6 +65,7 @@
         return _length;
     }

+#if 0
     Document& Hits::doc(int32_t n){
         HitDoc* hitDoc = getHitDoc(n);

@@ -87,7 +87,13 @@

         return *hitDoc->doc;
     }
+#endif

+   boost::python::object Hits::doc(const int32_t n) {
+       HitDoc* hitDoc = getHitDoc(n);
+       return searcher->doc(hitDoc->id);
+   }
+
     int32_t Hits::id (const int32_t n){
         return getHitDoc(n)->id;
     }
Index: BooleanQuery.h
```

```
===================================================================
--- BooleanQuery.h  (.../vendor/clucene-core-0.9.20/src/CLucene/search/BooleanQuery
    .h)  (revision 115)
+++ BooleanQuery.h  (.../branches/clucene-search/src/lucille/search/CLucene/search/
    BooleanQuery.h)  (revision 115)
@@ -54,8 +54,9 @@
       Weight* _createWeight(Searcher* searcher) {
          return _CLNEW BooleanWeight(searcher,&clauses,this);
       }
+   public:
108       BooleanQuery(const BooleanQuery& clone);
-   public:
+
     /** Constructs an empty boolean query. */
         BooleanQuery();

@@ -114,7 +115,7 @@

       Query* rewrite(CL_NS(index)::IndexReader* reader);
       Query* clone() const;
118 -     bool equals(Query* o) const;
+       bool equals(const Query* o) const;

       /** Prints a user-readable version of this query. */
       TCHAR* toString(const TCHAR* field) const;
Index: BooleanClause.h
===================================================================
--- BooleanClause.h (.../vendor/clucene-core-0.9.20/src/CLucene/search/
    BooleanClause.h) (revision 115)
+++ BooleanClause.h (.../branches/clucene-search/src/lucille/search/CLucene/search/
    BooleanClause.h) (revision 115)
@@ -31,11 +31,11 @@
128
       // If true, documents documents which <i>do not</i>
       //  match this sub-query will <i>not</i> match the boolean query.
-      bool required;
+      const bool required;

       // If true, documents documents which <i>do</i>
       //  match this sub-query will <i>not</i> match the boolean query.
-      bool prohibited;
+      const bool prohibited;
138
       bool deleteQuery;
```

```
Index: SearchHeader.h
===================================================================
--- SearchHeader.h  (.../vendor/clucene-core-0.9.20/src/CLucene/search/SearchHeader
    .h)  (revision 115)
+++ SearchHeader.h  (.../branches/clucene-search/src/lucille/search/CLucene/search/
    SearchHeader.h)  (revision 115)
@@ -11,10 +11,12 @@
 # pragma once
 #endif
148
+#include <boost/python.hpp>
+
 #include "CLucene/index/IndexReader.h"
 #include "CLucene/index/Term.h"
 #include "Filter.h"
-#include "CLucene/document/Document.h"
+//#include "CLucene/document/Document.h"
 #include "Sort.h"
 #include "CLucene/util/VoidList.h"
158  #include "Explanation.h"
@@ -133,7 +135,10 @@
     public:
         float_t score;
         int32_t id;
+#if 0
         CL_NS(document)::Document* doc;
+#endif
+       boost::python::object doc;

168         HitDoc* next;                       // in doubly-linked cache
         HitDoc* prev;                       // in doubly-linked cache
@@ -167,6 +172,7 @@
         /** Returns the total number of hits available in this set. */
         int32_t length() const;

+#if 0
         /** Returns the stored fields of the n<sup>th</sup> document in this set.
         <p>Documents are cached, so that repeated requests for the same element may
         return the same Document object.
178  @@ -174,6 +180,9 @@
         * @memory Memory belongs to the hits object. Don't delete the return value.
         */
         CL_NS(document)::Document& doc(const int32_t n);
```

```
+#endif
+
+       boost::python::object doc(const int32_t n);

        /** Returns the id for the nth document in this set. */
        int32_t id (const int32_t n);
@@ -248,13 +257,17 @@
       */
       virtual TopDocs* _search(Query* query, Filter* filter, const int32_t n) = 0;

+#if 0
       /** Expert: Returns the stored fields of document <code>i</code>.
       * Called by {@link HitCollector} implementations.
       * @see IndexReader#document(int32_t).
       */
       virtual bool doc(int32_t i, CL_NS(document)::Document* d) = 0;
       _CL_DEPRECATED( doc(i, document) ) CL_NS(document)::Document* doc(const
           int32_t i);
+#endif

+      virtual boost::python::object doc(const int32_t i);
+
       /** Expert: called to re-write queries into primitive queries. */
       virtual Query* rewrite(Query* query) = 0;

@@ -435,7 +448,7 @@
         */
         virtual TCHAR* toString(const TCHAR* field) const = 0;

-        virtual bool equals(Query* other) const = 0;
+        virtual bool equals(const Query* other) const = 0;
         virtual size_t hashCode() const = 0;

         /** Prints a query to a string. */
Index: TermScorer.cpp
===================================================================
--- TermScorer.cpp  (.../vendor/clucene-core-0.9.20/src/CLucene/search/TermScorer.
    cpp)  (revision 115)
+++ TermScorer.cpp  (.../branches/clucene-search/src/lucille/search/CLucene/search/
    TermScorer.cpp)  (revision 115)
@@ -15,7 +15,7 @@

    //TermScorer takes TermDocs and delets it when TermScorer is cleaned up
    TermScorer::TermScorer(Weight* w, CL_NS(index)::TermDocs* td,
```

```
-           Similarity* similarity,uint8_t* _norms):
+           Similarity* similarity, const uint8_t* _norms):
        Scorer(similarity),
        termDocs(td),
        norms(_norms),
Index: IndexSearcher.h
===================================================================
--- IndexSearcher.h (.../vendor/clucene-core-0.9.20/src/CLucene/search/
    IndexSearcher.h) (revision 115)
+++ IndexSearcher.h (.../branches/clucene-search/src/lucille/search/CLucene/search/
    IndexSearcher.h) (revision 115)
@@ -11,9 +11,11 @@
 # pragma once
 #endif

+#include <boost/python.hpp>
+
 #include "SearchHeader.h"
-#include "CLucene/store/Directory.h"
-#include "CLucene/document/Document.h"
+//#include "CLucene/store/Directory.h"
+//#include "CLucene/document/Document.h"
 #include "CLucene/index/IndexReader.h"
 #include "CLucene/index/Term.h"
 #include "CLucene/util/BitSet.h"
@@ -31,15 +33,11 @@
    bool readerOwner;

 public:
-   /// Creates a searcher searching the index in the named directory.
-   IndexSearcher(const char* path);
-
-    /// Creates a searcher searching the index in the specified directory.
-    IndexSearcher(CL_NS(store)::Directory* directory);
-
    /// Creates a searcher searching the provided index.
    IndexSearcher(CL_NS(index)::IndexReader* r);

+   IndexSearcher(boost::python::object r);
+
    ~IndexSearcher();

    /// Frees resources associated with this Searcher.
@@ -47,9 +45,13 @@
```

```
        int32_t docFreq(const CL_NS(index)::Term* term) const;

268 +#if 0
        bool doc(int32_t i, CL_NS(document)::Document* document);
        _CL_DEPRECATED( doc(i, document) ) CL_NS(document)::Document* doc(int32_t i);
    +#endif

    +   boost::python::object doc(const int32_t i);
    +
        int32_t maxDoc() const;

        TopDocs* _search(Query* query, Filter* filter, const int32_t nDocs);
278 Index: FieldCacheImpl.cpp
    ===================================================================
    --- FieldCacheImpl.cpp  (.../vendor/clucene-core-0.9.20/src/CLucene/search/
        FieldCacheImpl.cpp)  (revision 115)
    +++ FieldCacheImpl.cpp  (.../branches/clucene-search/src/lucille/search/CLucene/
        search/FieldCacheImpl.cpp)  (revision 115)
    @@ -5,6 +5,7 @@
     * the GNU Lesser General Public License, as specified in the COPYING file.
     ------------------------------------------------------------------------------*/
     #include "CLucene/StdHeader.h"
    +#include "CLucene/util/StringIntern.h"
     #include "FieldCacheImpl.h"
288
     CL_NS_USE(util)
    Index: TermQuery.h
    ===================================================================
    --- TermQuery.h (.../vendor/clucene-core-0.9.20/src/CLucene/search/TermQuery.h) (
        revision 115)
    +++ TermQuery.h (.../branches/clucene-search/src/lucille/search/CLucene/search/
        TermQuery.h) (revision 115)
    @@ -11,6 +11,8 @@
     # pragma once
     #endif

298 +#include <boost/python.hpp>
    +
     #include "SearchHeader.h"
     #include "Scorer.h"
     #include "CLucene/index/Term.h"
    @@ -55,22 +57,27 @@
```

```
          protected:
              Weight* _createWeight(Searcher* searcher);
    +     public:
308           TermQuery(const TermQuery& clone);
    -     public:
    +
              // Constructs a query for the term <code>t</code>.
              TermQuery(CL_NS(index)::Term* t);
              ~TermQuery();

    +         TermQuery(boost::python::object _term);
    +
              static const TCHAR* getClassName();
318           const TCHAR* getQueryName() const;

              //added by search highlighter
              CL_NS(index)::Term* getTerm(bool pointer=true) const;
    +
    +         CL_NS(index)::Term* _getTermPython() const;

              // Prints a user-readable version of this query.
              TCHAR* toString(const TCHAR* field) const;

328 -         bool equals(Query* other) const;
    +         bool equals(const Query* other) const;
              Query* clone() const;

              /** Returns a hash code value for this object.*/
    Index: BooleanQuery.cpp
    ===================================================================
    --- BooleanQuery.cpp    (.../vendor/clucene-core-0.9.20/src/CLucene/search/
        BooleanQuery.cpp)    (revision 115)
    +++ BooleanQuery.cpp    (.../branches/clucene-search/src/lucille/search/CLucene/
        search/BooleanQuery.cpp)    (revision 115)
    @@ -191,7 +191,7 @@
338           }

              /** Returns true iff <code>o</code> is equal to this. */
    -         bool BooleanQuery::equals(Query* o)const {
    +         bool BooleanQuery::equals(const Query* o)const {
                 if (!(o->instanceOf(BooleanQuery::getClassName())))
                    return false;
                 const BooleanQuery* other = (BooleanQuery*)o;
    Index: MultiSearcher.h
```

```
    ===================================================================
348 --- MultiSearcher.h (.../vendor/clucene-core-0.9.20/src/CLucene/search/
        MultiSearcher.h) (revision 115)
    +++ MultiSearcher.h (.../branches/clucene-search/src/lucille/search/CLucene/search/
        MultiSearcher.h) (revision 115)
    @@ -12,7 +12,7 @@
     #endif

     #include "SearchHeader.h"
    -#include "CLucene/document/Document.h"
    +//#include "CLucene/document/Document.h"
     #include "CLucene/index/Term.h"

358  CL_NS_DEF(search)
    @@ -54,9 +54,13 @@

          int32_t docFreq(const CL_NS(index)::Term* term) const ;

    +#if 0
           /** For use by {@link HitCollector} implementations. */
          bool doc(int32_t n, CL_NS(document)::Document* document);
    +#endif

368 +     boost::python::object doc(const int32_t n);
    +
           /** For use by {@link HitCollector} implementations to identify the
            * index of the sub-searcher that a particular hit came from. */
           int32_t searcherIndex(int32_t n) const;
    Index: SearchHeader.cpp
    ===================================================================
    --- SearchHeader.cpp    (.../vendor/clucene-core-0.9.20/src/CLucene/search/
        SearchHeader.cpp)    (revision 115)
    +++ SearchHeader.cpp    (.../branches/clucene-search/src/lucille/search/CLucene/
        search/SearchHeader.cpp)    (revision 115)
    @@ -12,13 +12,19 @@
378  CL_NS_USE(index)
     CL_NS_DEF(search)

    +#if 0
     CL_NS(document)::Document* Searchable::doc(const int32_t i){
         CL_NS(document)::Document* ret = _CLNEW CL_NS(document)::Document;
         if (!doc(i,ret) )
             _CLDELETE(ret);
         return ret;
```

```
 }
+#endif

+boost::python::object Searchable::doc(const int32_t i) {
+   throw false;
+}
+
 //static
 Query* Query::mergeBooleanQueries(Query** queries) {
     CL_NS(util)::CLVector<BooleanClause*> allClauses;
Index: IndexSearcher.cpp
===================================================================
--- IndexSearcher.cpp   (.../vendor/clucene-core-0.9.20/src/CLucene/search/
    IndexSearcher.cpp)   (revision 115)
+++ IndexSearcher.cpp   (.../branches/clucene-search/src/lucille/search/CLucene/
    search/IndexSearcher.cpp)   (revision 115)
@@ -10,8 +10,8 @@
 #include "SearchHeader.h"
 #include "Scorer.h"
 #include "FieldDocSortedHitQueue.h"
-#include "CLucene/store/Directory.h"
-#include "CLucene/document/Document.h"
+//#include "CLucene/store/Directory.h"
+//#include "CLucene/document/Document.h"
 #include "CLucene/index/IndexReader.h"
 #include "CLucene/index/Term.h"
 #include "CLucene/util/BitSet.h"
@@ -19,7 +19,7 @@

 CL_NS_USE(index)
 CL_NS_USE(util)
-CL_NS_USE(document)
+//CL_NS_USE(document)

 CL_NS_DEF(search)

@@ -102,30 +102,6 @@
     };


-  IndexSearcher::IndexSearcher(const char* path){
-  //Func - Constructor
-  //       Creates a searcher searching the index in the named directory.  */
-  //Pre  - path != NULL
```

```
-  //Post - The instance has been created
-
-      CND_PRECONDITION(path != NULL, "path is NULL");
-
-      reader = IndexReader::open(path);
-      readerOwner = true;
-  }
-
-  IndexSearcher::IndexSearcher(CL_NS(store)::Directory* directory){
438 -  //Func - Constructor
-  //       Creates a searcher searching the index in the specified directory.  */
-  //Pre  - path != NULL
-  //Post - The instance has been created
-
-      CND_PRECONDITION(directory != NULL, "directory is NULL");
-
-      reader = IndexReader::open(directory);
-      readerOwner = true;
-  }
448 -
     IndexSearcher::IndexSearcher(IndexReader* r){
     //Func - Constructor
     //       Creates a searcher searching the index with the provide IndexReader
    @@ -136,6 +112,11 @@
         readerOwner = false;
     }

+  IndexSearcher::IndexSearcher(boost::python::object r) {
+      reader = _CLNEW IndexReader(r);
458 +     readerOwner = true;
+  }
+
     IndexSearcher::~IndexSearcher(){
     //Func - Destructor
     //Pre  - true
    @@ -165,6 +146,7 @@
         return reader->docFreq(term);
     }

468 +#if 0
     _CL_DEPRECATED( doc(i, document) ) CL_NS(document)::Document* IndexSearcher::doc
         (int32_t i){
      CL_NS(document)::Document* ret = _CLNEW CL_NS(document)::Document;
      if (!doc(i,ret) )
```

```
@@ -183,7 +165,12 @@

        return reader->document(i,d);
    }
+#endif

+  boost::python::object IndexSearcher::doc(const int32_t i) {
+   return reader->document(i);
+  }
+
    // inherit javadoc
    int32_t IndexSearcher::maxDoc() const {
    //Func - Return total number of documents including the ones marked deleted
Index: Sort.cpp
===================================================================
--- Sort.cpp    (.../vendor/clucene-core-0.9.20/src/CLucene/search/Sort.cpp)    (
    revision 115)
+++ Sort.cpp    (.../branches/clucene-search/src/lucille/search/CLucene/search/Sort
    .cpp)    (revision 115)
@@ -5,6 +5,7 @@
 * the GNU Lesser General Public License, as specified in the COPYING file.
 ------------------------------------------------------------------------------*/
 #include "CLucene/StdHeader.h"
+#include "CLucene/util/StringBuffer.h"
 #include "Sort.h"
 #include "Compare.h"

Index: TermScorer.h
===================================================================
--- TermScorer.h    (.../vendor/clucene-core-0.9.20/src/CLucene/search/TermScorer.h
    )    (revision 115)
+++ TermScorer.h    (.../branches/clucene-search/src/lucille/search/CLucene/search/
    TermScorer.h)    (revision 115)
@@ -21,7 +21,7 @@
    class TermScorer: public Scorer {
    private:
        CL_NS(index)::TermDocs* termDocs;
-       uint8_t* norms;
+       const uint8_t* norms;
        Weight* weight;
        const float_t weightValue;
        int32_t _doc;
@@ -36,7 +36,7 @@
```

```
            //TermScorer takes TermDocs and delets it when TermScorer is cleaned up
            TermScorer(Weight* weight, CL_NS(index)::TermDocs* td,
-           Similarity* similarity, uint8_t* _norms);
+           Similarity* similarity, const uint8_t* _norms);

            ~TermScorer();
518
    Index: BooleanScorer.cpp
    ===================================================================
    --- BooleanScorer.cpp   (.../vendor/clucene-core-0.9.20/src/CLucene/search/
        BooleanScorer.cpp)   (revision 115)
    +++ BooleanScorer.cpp   (.../branches/clucene-search/src/lucille/search/CLucene/
        search/BooleanScorer.cpp)   (revision 115)
    @@ -5,6 +5,7 @@
     * the GNU Lesser General Public License, as specified in the COPYING file.
     ------------------------------------------------------------------------------*/
     #include "CLucene/StdHeader.h"
    +#include "CLucene/util/StringBuffer.h"
528  #include "BooleanScorer.h"

     #include "Scorer.h"
    Index: TermQuery.cpp
    ===================================================================
    --- TermQuery.cpp   (.../vendor/clucene-core-0.9.20/src/CLucene/search/TermQuery.
        cpp)   (revision 115)
    +++ TermQuery.cpp   (.../branches/clucene-search/src/lucille/search/CLucene/search/
        TermQuery.cpp)   (revision 115)
    @@ -32,6 +32,12 @@
            _CLDECDELETE(term);
        }
538
    +   TermQuery::TermQuery(boost::python::object _term) {
    +       Term *t = _CLNEW Term();
    +       t->set(_term);
    +       term = _CL_POINTER(t);
    +   }
    +
        Query* TermQuery::clone() const{
            return _CLNEW TermQuery(*this);
        }
548 @@ -55,8 +61,11 @@
            else
                return term;
        }
```

```
+
+   Term *TermQuery::_getTermPython() const {
+       return _CLNEW Term(*term);
+   }

-
    /** Prints a user-readable version of this query. */
    TCHAR* TermQuery::toString(const TCHAR* field) const{
        CL_NS(util)::StringBuffer buffer;
@@ -73,7 +82,7 @@
    }

    /** Returns true iff <code>o</code> is equal to this. */
-   bool TermQuery::equals(Query* other) const {
+   bool TermQuery::equals(const Query* other) const {
       if (!(other->instanceOf(TermQuery::getClassName())))
            return false;

@@ -180,7 +189,7 @@
        fieldExpl->addDetail(idfExpl);

        Explanation* fieldNormExpl = _CLNEW Explanation();
-       uint8_t* fieldNorms = reader->norms(field);
+       const uint8_t* fieldNorms = reader->norms(field);
        float_t fieldNorm =
            fieldNorms!=NULL ? Similarity::decodeNorm(fieldNorms[doc]) : 0.0f;
        fieldNormExpl->setValue(fieldNorm);
```

module.cpp: Boost.Python module definition

```
#include <boost/python.hpp>

#include "CLucene/StdHeader.h"
#include "CLucene/index/Term.h"
#include "SearchHeader.h"
#include "BooleanClause.h"
#include "BooleanQuery.h"
#include "TermQuery.h"
#include "IndexSearcher.h"

using namespace boost::python;
CL_NS_USE(index)
CL_NS_USE(search)
```

```
typedef Hits *(IndexSearcher::*search_hits_type)(Query *);

/* XXX should be auto_ptr<Query> but breaks? */
void BooleanQuery_add(BooleanQuery &bq, auto_ptr<TermQuery> q, bool required, bool
    prohibited) {
    bq.add(q.get(), required, prohibited);
    q.release();
}

BOOST_PYTHON_MODULE(search) {
    class_<Term>("_CLucene_Term", no_init)
        .add_property("field", &Term::_fieldPython)
        .add_property("text", &Term::_textPython)
        ;
    class_<Query, boost::noncopyable>("Query", no_init);
    class_<BooleanQuery, auto_ptr<BooleanQuery>, bases<Query> >("BooleanQuery")
        .def("add", &BooleanQuery_add);
    class_<TermQuery, auto_ptr<TermQuery>, bases<Query> >("TermQuery", init<object
        >())
        .def("term", &TermQuery::_getTermPython,
                return_value_policy<manage_new_object>());
    class_<Hits>("Hits", no_init)
        .def("length", &Hits::length)
        .def("doc", &Hits::doc)
        .def("id", &Hits::id)
        .def("score", &Hits::score)
        ;
    class_<IndexSearcher>("IndexSearcher", init<object>())
        .def("search", search_hits_type(&IndexSearcher::search),
                return_value_policy<manage_new_object>())
        .def("close", &IndexSearcher::close)
        ;
}
```