

CLucene (C++)

Contributed by BEN VAN KLINKEN and ITAMAR SYN-HERSHKO

An excerpt from Manning's *Lucene in Action, Second Edition*
by Mike McCandless, Erik Hatcher,
and Otis Gospodnetić

CLucene is an open source native port of Lucene to C++, created by Ben van Klinken in 2003. Since then, many other developers have contributed to the project. The library's API and index file format are guaranteed to match those of the Java Lucene version it is based on. Table 1 shows its current status.

Port feature	Port status
Port type	Native port
Programming languages	C++
Website	http://clucene.sourceforge.net/
Development status	Stable
Activity	Active development, active users
Last stable release	0.9.21b
Matching Lucene release	1.9.1
Compatible index format	Yes, 1.9.1
Compatible APIs	Yes
License	LGPL or Apache License 2.0

Table 1
CLucene summary

In its latest stable release, CLucene conforms to Lucene 1.9.1's API and index format, but ongoing development is active toward fixing issues and supporting more recent Lucene releases. As of this writing, development is proceeding on a source code branch toward full compatibility with Lucene's 2.3.2 release. Despite being officially marked unstable, the 2.3.2 branch seems quite stable and is already commonly used, although the APIs are still likely to change.

Adobe and Nero are believed to use CLucene in their products, as do other well-known open source projects like Strigi, [ht://Dig](http://Dig), and kio-clucene.

Motivation

Many companies and developers use C/C++ exclusively and can't take advantage of Lucene because it requires Java. CLucene offers the benefits of the Lucene world, while allowing those companies and developers to keep with the platforms and development tools they are most familiar with.

C++ developers are the main audience of CLucene. Since it's written in native code and has no prerequisites, it's also fairly easy to use the library from various high-level or scripting languages. Thanks to its flexible build system, native code, and small memory footprint, CLucene can also be used on embedded systems and mobile devices, where resources are tight and a JVM is usually not an option.

The project also aims to be attractive to people who like to use Lucene but want to increase performance or remove the overhead of using a JVM. Although the Java platform is constantly improving, basic operations like file handling and memory management will always be faster for C++ compiled code, since no underlying framework or garbage collection processes are involved. CLucene is guaranteed to provide better performance, even without the periodic code optimizations by its core team.

Although no current benchmarks are available to show this, those made with previous versions of Lucene and CLucene showed CLucene performed 5–10 times better than an equivalent version of Lucene, in terms of memory usage and execution speeds of indexing and searching operations. Both Lucene and CLucene have changed substantially since then.

API and index compatibility

The CLucene API is similar to Lucene's; code written in Java can be converted to C++ fairly easily. The drawback is that CLucene doesn't follow the generally accepted C++ coding standards. But due to the number of classes that would have to be redesigned and the difficulty it will pose for the process of keeping up with the original Lucene project, CLucene continues to follow a "Javaesque" coding standard. This approach also allows much of the user's code to be converted using macros and scripts.

Thanks to a full index-format compatibility, indexes built with Lucene are also searchable using CLucene and vice versa, as long as their index format version is supported by both. For example, as of this writing CLucene can read and write to indexes created by Lucene 2.3.2, but it will not work with indexes created or merged by Lucene 3+. Because backward compatibility is preserved in Lucene, even the most recent versions of Lucene can read indexes built with any version of CLucene, and those will still be readable by CLucene as long as they are not being written to by a more recent Lucene version.

Listing 1 shows a command-line program to perform basic indexing and searching. This program first indexes several documents with a single contents field. Following that, it runs a few searches against the generated in-memory index and prints the search results for each query.

Listing 1 Using CLucene's IndexWriter and IndexSearcher API

```

#include "CLucene.h"

using namespace lucene::analysis;
using namespace lucene::index;
using namespace lucene::document;
using namespace lucene::queryParser;
using namespace lucene::search;
using namespace lucene::store;

const TCHAR* docs[] = {
    _T("a b c d e"),
    _T("a b c d e a b c d e"),
    _T("a b c d e f g h i j"),
    _T("a c e"),
    _T("e c a"),
    _T("a c e a c e"),
    _T("a c e a b c"),
    NULL
};

const TCHAR* queries[] = {
    _T("a b"),
    _T("\"a b\""),
    _T("\"a b c\""),
    _T("a c"),
    _T("\"a c\""),
    _T("\"a c e\""),
    NULL
};

int main( int32_t, char** argv )
{
    SimpleAnalyzer analyzer;

    try {
        Directory* dir = new RAMDirectory();
        IndexWriter* writer = new IndexWriter(dir, &analyzer, true);

        Document doc;
        for (int j = 0; docs[j] != NULL; ++j) {
            doc.add( *_CLNEW Field(_T("contents"),
                                docs[j],
                                Field::STORE_YES |
                                Field::INDEX_TOKENIZED) );
            writer->addDocument(&doc);
            doc.clear();
        }

        writer->close();
        delete writer;

        IndexReader* reader = IndexReader::open(dir);
        IndexSearcher searcher(reader);
    }
}

```

Index these documents

Run these searches

Initialize analyzer on stack

Reuse Document instance

Index document

```

QueryParser parser(_T("contents"), &analyzer);
parser.setPhraseSlop(4);

Hits* hits = NULL;

for (int j = 0; queries[j] != NULL; ++j)
{
    Query* query = parser.parse(queries[j]);
    const wchar_t* qryInfo = query->toString(_T("contents"));
    _tprintf(_T("Query: %s\n"), qryInfo);
    delete[] qryInfo;

    hits = searcher.search(query);
    _tprintf(_T("%d total results\n"),
            hits->length());
    for (size_t i=0; i < hits->length() && i<10; i++) {
        Document* d = &hits->doc(i);
        _tprintf(_T("#%d. %s (score: %f)\n"),
                i, d->get(_T("contents")),
                hits->score(i));
    }

    delete hits;
    delete query;
}

searcher.close(); reader->close(); delete reader;
dir->close(); delete dir;

} catch (CLuceneError& e) {
    _tprintf(_T(" caught a exception: %s\n"), e.twhat());
} catch (...){
    _tprintf(_T(" caught an unknown exception\n"));
}
}

```

Parse
query

Run search;
print results

Supported platforms

Initially developed in Microsoft Visual Studio, CLucene also compiles in GCC, MinGW32, and the Borland C++ compiler. In addition to the Microsoft Windows platform, it has been successfully built on various Linux distributions (Red Hat, Ubuntu, and more), FreeBSD, Mac OS X, and Debian. The code supports both 32- and 64-bit versions of these platforms.

Today, CLucene comes with CMake build scripts which simplify the build process, and allow it to be run on almost every platform. Both Unicode and non-Unicode builds are supported through it.

The CLucene team has made use of SourceForge's multiplatform compile farm to ensure that CLucene compiles and runs on as many platforms as possible. However, SourceForge has now closed its compile farm, so most cross-platform testing is done by contributors having physical access to various machines (even the rare ones), and by using virtual machines.

Current and future work

As part of the effort toward compatibility with Java Lucene, the distribution package of CLucene includes many of the same components as Lucene, such as tests, contrib folder, and a demo application. This is the case also with the development repositories. Unfortunately, Lucene's rapid growth makes it very hard to keep up with it in real time; therefore many classes and tests may be missing.

CLucene once had several wrappers that allowed it to be used with other programming languages, such as Perl, Python, .NET, and PHP. Most were made for previous versions of the library, and haven't been updated in some time. It's possible to bring them up to speed, or to use tools like SWIG to create one simple interface for many languages at once, should one need to use them again.

Following a decision made during early development, no external libraries were incorporated for string handling, threading, and reference counting. The core team has begun to replace the custom code and macros used for those operations with Boost's C++ libraries. This update will make CLucene much more robust, and allow its developers to focus solely on porting more Lucene code, instead of worrying about platform-specific issues. Also, introduction of concepts like smart pointers will make building wrappers much easier.