Application Developer's Guide

Japanese Morphological Analyzer

v2.0

Jun Jiang

September 20, 2010

Contents

Chapter 1

How to build and link with the library

1.1 Build the library

CMake[1] is a prerequisite to the library. We use it to generate Makefile for each platform.

1.1.1 Linux platform

On Linux platform, we could build like below.

```
$ cd build
$ cmake ../source
$ make
```

After the project is built, the library target libjma.a is created in directory lib, and the executables in directory bin are created for demo and test.

1.1.2 Win32 platform

On Win32 platform, please first check whether Visual Studio 9 2008 is installed beforehand. Below is an example to generate MSVC project file JMA.sln, please open it to build inside MSVC IDE.

```
$ cd build
$ cmake -G "Visual Studio 9 2008" ../source
```

[1] http://www.cmake.org

You could also use other Visual Studio version, in which case you need to replace Visual Studio 9 2008 to your version in below example.

1.1.3 AIX platform

On AIX platform, we could build using XLC compiler like below.

```
$ cd build
$ export CC=xlC_r CXX=xlC_r OBJECT_MODE=64
$ cmake -DCMAKE_COMPILER_IS_XLC=1 ../source
$ make
```

1.2 Link with the library

To link with the library, the user application needs to include header files in directory include, and link the library files libjma.a in directory lib.

Below is an example to compile user application test.cpp on Linux platform:

```
$ export JMA_PATH=path_of_jma_project
$ g++ -o test -I$JMA_PATH/include test.cpp $JMA_PATH/lib/libjma.a
```

Chapter 2

Get the dictionary

The library runs by loading dictionary files at run time. There are two types of dictionary, system dictionary and user dictionary.

2.1 System dictionary

At run time, the library loads the system dictionary in binary format. They could be downloaded like below[1].

```
$ rsync -rvP dev@izenesoft.com:jma-data/db/ipadic .
```

Directory ipadic contains four sub-directories. In src directory, it contains source files to compile into binary format. The other three directories contain the compiled binary format for each character encoding type. Directory bin_eucjp is for EUC-JP encoding, bin_sjis for SHIFT-JIS encoding, and bin_utf8 for UTF-8 encoding.

2.2 User dictionary

User dictionary file is in text format, which contains nouns defined by user. The sample user dictionary file could be downloaded like below.

```
$ rsync -rvP dev@izenesoft.com:jma-data/db/userdic .
```

[1] If your download access is denied, please follow the instructions in https://ssl.izenesoft.cn/projects/izenesoft/wiki/Resources

Directory userdic contains three sample files. Each is encoded in one character encoding type of EUC-JP, SHIFT-JIS and UTF-8. The grammar of user dictionary is defined in the comment section of these sample files.

Chapter 3

How to run the demo

To run the demo bin/jma_run, please make sure the project is built and system dictionary is downloaded (see previous two chapters).

Below is the demo general usage:

```
$ cd bin
$ ./jma_run --sentence N-best [--dict DICT_PATH]
$ ./jma_run --string [--dict DICT_PATH]
$ ./jma_run --stream INPUT OUTPUT [--dict DICT_PATH]
```

The DICT_PATH is the path of binary system dictionary, such like ipadic/bin_utf8. Below are the details to run each demo.

3.1 Analyze n-best results

To analyze 5-best results of a sentence string from standard input.

```
$ ./jma_run --sentence 5 --dict ipadic/bin_utf8
```

To exit the loop in the above demos, please press CTRL-D.

3.2 Analyze a string

To analyze a paragraph string from standard input.

```
$ ./jma_run --string --dict ipadic/bin_utf8
```

3.3 Analyze a file

To analyze file from INPUT to OUTPUT. Please note that the encoding type of INPUT file should be the same to the encoding type of system dictionary in DICT_PATH.

```
$ ./jma_run --stream INPUT OUTPUT --dict ipadic/bin_utf8
```

Chapter 4

Interface description

The C++ API of the library is described below:

Class JMA_Factory creates instances for JMA, its methods below create instances of JMA_Factory, Analyzer and Knowledge, which are used in the morphological analysis.

```
static JMA_Factory* instance();
virtual Analyzer* createAnalyzer() = 0;
virtual Knowledge* createKnowledge() = 0;
```

Class Knowledge manages the linguistic information, its methods below load files of system dictionary, user dictionary, stop-word dictionary and set part-of-speech tags for keywords to extract.

```
void setSystemDict(const char* dirPath);
void addUserDict(const char* fileName, EncodeType type = ENCODE_TYPE_UTF8);
virtual int loadDict() = 0;
virtual int loadStopWordDict(const char* fileName) = 0;
virtual int setKeywordPOS(const std::vector<std::string>& posVec) = 0;

EncodeType getEncodeType() const;
virtual int encodeSystemDict(const char* txtFileName, const char* binFileName) = 0;
```

After dictionary files are loaded, we could get its encoding type by Knowledge::getEncodeType(). Knowledge::encodeSystemDict() is the interface of encoding system dictionary from text format to binary format.

Class Analyzer executes the morphological analysis, its methods below execute the morphological analysis based on a sentence, a paragraph or a file separately. Analyzer::splitSentence() splits a paragraph into sentences. Analyzer::convertCharacters() transforms characters between different types, such as Katakana and Hiragana.

```
virtual int setKnowledge(Knowledge* pKnowledge) = 0;
void setOption(OptionType nOption, double nValue);

virtual int runWithSentence(Sentence& sentence) = 0;
virtual const char* runWithString(const char* inStr) = 0;
virtual int runWithStream(const char* inFileName, const char* outFileName) = 0;

virtual void splitSentence(const char* paragraph, std::vector<Sentence>& sentences) = 0;
virtual std::string convertCharacters(const char* str) const = 0;
```

Class Sentence saves the analysis results of Analyzer::runWithSentence(), so that the n-best or one-best results could be accessed. And you could get the string of lexicon, part-of-speech tag, basic form, pronunciation form and normalization form in each candidate result.

```
void setString(const char* pString);
const char* getString(void) const;
int getListSize(void) const;
int getCount(int nPos) const;
const char* getLexicon(int nPos, int nIdx) const;
int getPOS(int nPos, int nIdx) const;
const char* getStrPOS(int nPos, int nIdx) const;
const char* getBaseForm(int nPos, int nIdx) const;
const char* getReadForm(int nPos, int nIdx) const;
const char* getNormForm(int nPos, int nIdx) const;
double getScore(int nPos) const;
int getOneBestIndex(void) const;
```

Chapter 5

How to use the interface

In general, the interface could be used following below steps:

1. Include the header files in directory include.

```
#include "ijma.h"
```

2. Use the library name space.

```
using namespace jma;
```

3. Call the interface and handle the result.

In the example below, the return value of some functions are not handled for simplicity. In your using, please properly handle those return values in case of failure. The interface details could be available either in document docs/html/namespacejma.html, or the comments in the header files of directory include.

```
// create instances
JMA_Factory* factory = JMA_Factory::instance();
Analyzer* analyzer = factory->createAnalyzer();
Knowledge* knowledge = factory->createKnowledge();

// set the system dictionary path
knowledge->setSystemDict("ipadic/bin_eucjp");
// (optional) add the paths of multiple user dictionaries
knowledge->addUserDict("userdic/user_noun.utf8", Knowledge::ENCODE_TYPE_UTF8);
knowledge->addUserDict("userdic/user_noun.eucjp", Knowledge::ENCODE_TYPE_EUCJP);
knowledge->addUserDict("userdic/user_noun.sjis", Knowledge::ENCODE_TYPE_SJIS);
// load system and user dictionary files
knowledge->loadDict();

// (optional) load stop word dictionary
knowledge->loadStopWordDict("...");
```

```
// (optional) configure analysis option if necessary
analyzer->setOption(Analyzer::OPTION_TYPE_***, ...);

// set knowledge
analyzer->setKnowledge(knowledge);

// analyze a sentence
Sentence s("...");
analyzer->runWithSentence(s);

// analyze a paragraph
const char* result = analyzer->runWithString("...");

// analyze a file
analyzer->runWithStream("...", "...");

// destroy instances
delete knowledge;
delete analyzer;
```

Below are some guidelines which might be helpful in your using the interface.

5.1 How to get n-best results

First you might need to set n-best option value to specify how many candidates you want. In below example, it would get 5 best candidates. If this option is not set, JMA would do one-best analysis.

```
analyzer->setOption(Analyzer::OPTION_TYPE_NBEST, 5);
```

Then you might need to split a paragraph string into each sentence. In below example, it splits a paragraph string into a sentence vector, and then calls Analyzer::runWithSentence() to analyze each sentence.

```
vector<Sentence> sentVec;
analyzer->splitSentence("...", sentVec);
for(size_t i=0; i<sentVec.size(); ++i)
{
    analyzer->runWithSentence(sentVec[i]);
    ...
}
```

After Analyzer::runWithSentence() is called, you could get n-best results like below.

```
Sentence s("...");
analyzer->runWithSentence(s);
```

```
// get one-best result
int i= s.getOneBestIndex();
...

// get n-best results
for(int i=0; i<s.getListSize(); ++i)
{
    // each candidate result
    for(int j=0; j<s.getCount(i); ++j)
    {
        const char* lexicon = s.getLexicon(i, j);
        const char* pos = s.getStrPOS(i, j);
        const char* baseForm = s.getBaseForm(i, j);
        const char* readForm = s.getReadForm(i, j);
        const char* normForm = s.getNormForm(i, j);
        ...
    }
    double score = s.getScore(i);
    ...
}
```

5.2 How to control part-of-speech tag output

If you need not part-of-speech tag printed in Analyzer::runWithString() or Analyzer::runWithStream(), you could set option like below.

```
analyzer->setOption(Analyzer::OPTION_TYPE_POS_TAGGING, 0);
```

In Analyzer::runWithSentence(), Analyzer::runWithString() and Analyzer::runWithStream(), if you need part-of-speech tag in alphabet format such as “NC-G”, instead of the default format such as “名詞,一般”, you could set option like below.

```
analyzer->setOption(Analyzer::OPTION_TYPE_POS_FORMAT_ALPHABET, 1);
```

5.3 How to extract keywords

You could specify the part-of-speech tags of keywords like below, so that only the keywords would be printed in Analyzer::runWithSentence(), Analyzer::runWithString() and Analyzer::runWithStream(). Please note that you should specify the part-of-speech tag in alphabet format.

```
vector<string> posVec;
posVec.push_back("NC-G");
posVec.push_back("V-I");
knowledge->setKeywordPOS(posVec);
```

5.4 How to combine words into compounds

In the system dictionary directory, the file compound.def defines the rules to combine a sequence of words into compound word. The rules are defined using part-of-speech tags in alphabet format, and users could edit these rules directly.

If you need not combine words into compounds, you could set option like below.

```
analyzer->setOption(Analyzer::OPTION_TYPE_COMPOUND_MORPHOLOGY, 0);
```

5.5 How to decompose user defined nouns

In user dictionary, user could define decomposition pattern, such as "本田 総一郎 2,3", which means this user defined noun could be decomposed into two words "本田" (2 characters) and "総一郎" (3 characters).

To enable decomposing user defined nouns, you need to set option like below.

```
analyzer->setOption(Analyzer::OPTION_TYPE_DECOMPOSE_USER_NOUN, 1);
```

5.6 How to transform characters

Analyzer::convertCharacters() transforms characters between all kinds of types. These types are configured using options Analyzer::OPTION_TYPE_CONVERT_TO_*, which are explained below.

To convert from Japanese Katanaka characters to their Hiragana equivalents, such as "タ" to "た":

```
analyzer->setOption(Analyzer::OPTION_TYPE_CONVERT_TO_HIRAGANA, 1);
```

To convert from Japanese Hiragana characters to their Katanaka equivalents, such as "た" to "タ":

```
analyzer->setOption(Analyzer::OPTION_TYPE_CONVERT_TO_KATAKANA, 1);
```

To convert digits and alphabets to their half width equivalents, such as "A" to "A":

```
analyzer->setOption(Analyzer::OPTION_TYPE_CONVERT_TO_HALF_WIDTH, 1);
```

To convert digits and alphabets to their full width equivalents, such as “A” to “A”:

```
analyzer->setOption(Analyzer::OPTION_TYPE_CONVERT_TO_FULL_WIDTH, 1);
```

To convert alphabets to their lower case equivalents, such as “A” to “a”:

```
analyzer->setOption(Analyzer::OPTION_TYPE_CONVERT_TO_LOWER_CASE, 1);
```

To convert alphabets to their upper case equivalents, such as “a” to “A”:

```
analyzer->setOption(Analyzer::OPTION_TYPE_CONVERT_TO_UPPER_CASE, 1);
```

Appendix A

Appendix - Part-of-Speech Tagset

Table A.1: IPA Dictionary POS Tags

Tag in Alphabet	Tag in Japanese	Description
O	その他,間投	other
F	フィラー	filler
I	感動詞	interjection
S-A	記号,アルファベット	alphabet symbol
S-G	記号,一般	general symbol
PUNCT-L	記号,括弧開	left parenthesis
PUNCT-R	記号,括弧閉	right parenthesis
EOS	記号,句点	sentence-final punctuation
S-S	記号,空白	space character
PUNCT-C	記号,読点	comma punctuation
AJ-I	形容詞,自立	independent adjective
AJ-S	形容詞,接尾	adjective as suffix
AJ-D	形容詞,非自立	dependent adjective
PL-G	助詞,格助詞,一般	general particle
PL-R	助詞,格助詞,引用	particle for reference
PL-C	助詞,格助詞,連語	particle for connection
PL-A	助詞,係助詞	particle for accompany
PL-E	助詞,終助詞	particle as end of sentence
PL-J	助詞,接続助詞	particle for conjunction
PL-O	助詞,特殊	special particle
PL-DS	助詞,副詞化	particle as adverb suffix
PL-D	助詞,副助詞	particle as adverb

Continued on next page

Table A.1 – continued from previous page

Tag in Alphabet	**Tag in Japanese**	**Description**
PL-K	助詞,副助詞 / 並立助詞 / 終助詞	particle as adverb, compound, or sentence end
PL-P	助詞,並立助詞	particle for compound
PL-N	助詞,連体化	particle after noun
AUV	助動詞	auxiliary verb
J	接続詞	conjunction
P-A	接頭詞,形容詞接続	prefix before adjective
P-NN	接頭詞,数接続	prefix before number
P-V	接頭詞,動詞接続	prefix before verb
P-N	接頭詞,名詞接続	prefix before noun
V-I	動詞,自立	independent verb
V-S	動詞,接尾	verb as suffix
V-D	動詞,非自立	dependent verb
D-G	副詞,一般	general adverb
D-P	副詞,助詞類接続	adverb as particle conjunction
N-VS	名詞,サ変接続	noun as suru-verb
N-NE	名詞,ナイ形容詞語幹	noun as adjective before negative auxiliary verb
NC-G	名詞,一般	common noun
N-R	名詞,引用文字列	noun for reference
N-AJV	名詞,形容動詞語幹	noun as adjective verb
NP-G	名詞,固有名詞,一般	general proper noun
NP-H	名詞,固有名詞,人名,一般	person name
NP-S	名詞,固有名詞,人名,姓	surname
NP-GN	名詞,固有名詞,人名,名	given name
NP-O	名詞,固有名詞,組織	organization
NP-P	名詞,固有名詞,地域,一般	place name
NP-C	名詞,固有名詞,地域,国	country name
NN	名詞,数	numeral noun
NJ	名詞,接続詞的	conjunction noun
NS-SN	名詞,接尾,サ変接続	suru-noun as suffix
NS-G	名詞,接尾,一般	general noun suffix
NS-AJV	名詞,接尾,形容動詞語幹	noun suffix as adjective verb
NU	名詞,接尾,助数詞	measure word
NS-AUV	名詞,接尾,助動詞語幹	noun suffix as auxiliary verb
NS-H	名詞,接尾,人名	person name suffix
NS-P	名詞,接尾,地域	place name suffix
NS-S	名詞,接尾,特殊	special noun suffix
NS-D	名詞,接尾,副詞可能	noun suffix as adverb

Continued on next page

Table A.1 – continued from previous page

Tag in Alphabet	**Tag in Japanese**	**Description**
NR-G	名詞,代名詞,一般	pronoun
NR-A	名詞,代名詞,縮約	abbreviation pronoun
N-VD	名詞,動詞非自立的	noun as dependent verb
N-AUV	名詞,特殊,助動詞語幹	special noun as auxiliary verb
ND-G	名詞,非自立,一般	general dependent noun
AN	名詞,非自立,形容動詞語幹	adjectival noun
ND-AUV	名詞,非自立,助動詞語幹	dependent noun as auxiliary verb
ND-D	名詞,非自立,副詞可能	dependent noun as adverb
N-D	名詞,副詞可能	noun as adverb
AA	連体詞	adnominal adjective
N-USER	名詞,ユーザ	user defined noun