SoftCorporation LLC.

# Suggester User Manual

Version 1.1

# Table of Contents

# User Manual

## 1. Introduction

This manual is for the IT staff interested in incorporating the Suggester software into their proprietary systems.

The document contains technical information, which is required to use the Suggester software. It includes guidance for configuring and deploying the Suggester, as well the Suggester functional specification.

### 1.1 Version

The information in this book is current as of Suggester Advanced version 1.2.0.

### 1.2 Definitions, Acronyms and Abbreviations:

| Term | Definition |
|---|---|
| Suggester | The Suggester java library |
| API | Application Programming Interface |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |

## 2. Technical Support

SoftCorporation LLC Technical Support exists to provide you with accurate resolutions to difficulties relating to using SoftCorporation software products.

You can contact Technical Support using email: support@softcorporation.com

It is recommended that you periodically check the SoftCorporation LLC web site:
http://www.softcorporation.com/products/suggester/advanced for the existence of updates to this product and latest documentation.

# 3. Overview of the Suggester Software

This chapter provides an introduction to the concepts and ideas behind the Suggester.

These topics are covered:

- Introducing the Suggester
- Suggester Features

## 3.1 Introducing the Suggester

The Suggester Library is a set of Java classes, providing suggestions for unknown (misspelt) words based on custom dictionary.
System administrator can create a list of preferred words and assign higher weight to the list or to the individual word.
The Suggester includes high speed suggestion engine, based on fast edit-distance calculation algorithm enhanced with Lawrence Philips Metaphone algorithm and private fuzzy-matching algorithm.
In basic implementation the Suggester can serve as a spellchecker.

## 3.2 Suggester Features

**High dictionary compression:**
The word dictionary is compressed not only on a hard drive, but also in virtual memory. Basic UK English dictionary contains about 57000 words and has a size about 90K. Full English dictionary contains about 200,000 words (including names, abbreviations, geographic places, etc.) and it takes 236Kb file on a hard drive and about 2Mb space in memory. Other languages are compressed even better. For example, full Russian dictionary contains more than 1,300,000 words (including variants) and it takes 315Kb file on a hard drive and again about 2Mb space in memory. Comparing original word list file in UTF-8 format with size more than 30Mb, the compressed size is close to 1% of original size.

**High dictionary search and suggestion selection speed:**
Dictionary case dependent / independent look-up takes about 0.002 / 0.005 ms per word, which comes to speed about 500,000 / 200,000 words per second. Suggestions search speed averages about 40 ms per set of suggestions for each unknown word on Pentium M 1.4Gz.

**Portability:**
The Suggester software entirely written in Java 1.2. Runs on any Java platform: Windows, Mac OS, UNIX and Linux. Tested with JRE 1.2, 1.3, 1.4, 1.5, 1.6.

**Dictionary retains original word list:**
The dictionary internal structure supports UTF-8 encoding and keeps all original words in a case sensitive format.

## 3.3 Command-line Tools

The Suggester comes with the command-line tools, which allow creating compiled dictionaries from the word or phrase lists, as well as creating such lists from plain text, log files or other input text.

**Index Builder:**
The Index Builder creates compressed index from your word / phrases list.

**Collocations Extractor:**
The Collocations Extractor creates a list of word collocations extracted from your text(s).

## 3.4 Application Programming Interface (API)

The Suggester API allows you to interact with the Suggester software. Java 2 and versions above are supported. For more information on Java API, see the Java API section.

## 4. Index Builder

The Index Builder creates compressed index from your word / phrase list. You can specify one file or directory of files containing word / phrase lists. We define a word as a text without spaces, and the phrase is a sequence of words separated by spaces. Words and Phrases may have an integer, which represents the weight of a word / phrase and is separated by "tab" character from the text. Each line in input file defines one word or one phrase.

At present time the Suggester has Basic and Advanced Index Builder. Basic Index Builder can process only single input file, while Advanced Index Builder can process directories of files. Basic Index Builder does not support weighting of individual words / phrases.

### 4.1 Index Builder command line parameters

The Index Builder accepts following parameters in command line:

**-input filename[,filename]** input word file(s) (mandatory)

The value **filename** can be a single file or comma separated (no spaces) list of files, or a directory containing word files. The directory can be used only with AdvancedIndex Builder.

**-output filename** output index file (mandatory)

The value **filename** is the name of a file containing compiled dictionary.
IMPORTANT: You cannot change the name of a file, when the dictionary is compiled. To change the file name you need to recompile the dictionary using another file name.

**-name indexname** index name in output index file

The value **indexname** is a name of the dictionary. You can use this option of you need a name of the file and a name of the dictionary to be different. By default the file name and the dictionary name are the same.

**-encoding value** input word list file character encoding.

The value is the standard Java encoding (default encoding is UTF-8). If you have multiple input files, all of them should have the same character encoding.

**-wordlist filename** output sorted word list file in UTF-8 encoding

The value **filename** is a name of the file, where the Suggester will print all recognized words (n-grams) in the index.

**-language value** index language, value as ISO 639: 2-letter code

The value identifies the language, for which the index is optimized. Contact SoftCorporation LLC for the list of supported languages.

**-license value** license information

The value contains a license code.

**-info value** the index additional information

The value can contain additional information useful to identify the index.

| | |
|---|---|
| **-sorted** | input word list is already sored and tokenized |

This option can be used only with BasicIndexBuilder. The input word list must be sorted and tokenized. Use this option if your computer has small memory size, but you need to compile large word list.

| | |
|---|---|
| **-weighted** | each phrase or word has weight separated from phrase by tab |

This option can be used only with AdvancedIndexBuilder. Each word or n-gram will have a weight, which will be used at the sorting phase. For example, the number of occurrences of the word in a text can be used as a weight. Note: Due to compression, the weight is stored and used as not exact value, but an approximation, which means that close weights can be stored as the same value.

| | |
|---|---|
| **-tokenize** | separate words on each line in input file |

This option will break each word as a separate entry. It cannot be used with weights.

| | |
|---|---|
| **-verbose** | display progress information (on standard output) |

This option will print progress information on a screen, which can be redirected to a file for logging or used for debugging purpose.

## 4.2 Input file(s) format

The Index Builder takes as input a single text file, or a list of files or a directory containing text files. File(s) can be presented in following formats:

### 4.2.1. Simple word list.

The word list, where each word separated by the line break.
Example:

```
abroad
abrogate
abrupt
abscess
abscissa
abscissae
absence
```

The list can be sorted or unsorted, with duplicates or without, however sorted list without duplicates can give significant advantage in memory reduction.

### 4.2.2. Weighted word list.

The word list, where each word separated by the line break and each word containing a number associated with the weight of this word. The number is separated from the word by tab character. For example, the number can be a frequency or average number of occurrences of this word in the text.
Example:

```
the	100
```

| | |
|---|---|
| of | 98 |
| and | 94 |

The list can be sorted or unsorted. If the same word is repeated several times, the total number for this word will be a sum of numbers of each occurrence.

#### 4.2.3. Weighted word collocations or n-grams

The n-gram list is a file structure, where each line contains a sequence of words separated by the space character. Each line may contain an integer number associated with the weight of this n-gram. The weight number is separated from words by tab character.
Example:

| | |
|---|---|
| Serve as the incoming | 92 |
| Serve as the incubator | 99 |
| Serve as the independent | 794 |
| Serve as the index | 223 |
| Serve as the indication | 72 |
| Serve as the indicator | 120 |
| Serve as the indicators | 45 |

The list can be sorted or unsorted. The list also can be used without weights. If the same n-gram is repeated several times, the total number for this n-gram n-gram will be a summary of each occurrence.

## 4.3 Index Builder usage examples

1. Create index from single word file in UTF-8 character encoding and save output to a file index.zip:

```
java -mx128m com.softcorporation.suggester.index.BasicIndexBuilder -input words.txt -output index.zip
```

2. Create index from two ISO Latin 1 word files, save output to a file index.zip and print the list of recognized words to a file list.txt:

```
java -mx128m com.softcorporation.suggester.index.BasicIndexBuilder -input words1.txt,words2.txt -output index.zip -wordlist list.txt -encoding ISO8859_1
```

3. Create index from all files located in directory /opt/suggester/words/test/ with UTF-8 character encoding, assign weight to each phrase, save output to a file index.jar and print progress on the screen:

```
java –mx2048m com.softcorporation.suggester.index.AdvancedIndexBuilder -input /opt/suggester/words/test -output index.jar -encoding UTF-8 -weighted -verbose
```

## 4.4 Index Builder limitations

The Index Builder creates the index in random access memory, which means the memory should be large enough to fit all words and n-grams in non-compressed form. On average 2 GB of memory will be consumed to process about 200 Mb of n-grams.

# 5. Collocations Extractor

The Collocations Extractor creates a list of collocated terms or n-grams, which can be used to make a list of suggestions based on your own text(s). You can specify one file or directory of files containing input text. The input file(s) should contain a plain text, where one line is one sentence, or name of an item, or title of an article, etc. Output file is a list of extracted words / phrases with an integer, representing the number of how many times this word / phrase was repeated in the text, and the weight is separated for the n-gram by “tab” character (09). Each line in output file contains one word or one n-gram (phrase).

## 5.1 Colocations Extractor command line parameters

The Colocations Extractor accepts following parameters in command line:

**-input filename[,filename]** input text file(s) (mandatory)

The value **filename** can be a single file or comma separated (no spaces) list of files, or a directory containing text files.

**-output filename** output word list file (mandatory)

The value **filename** is the name of a file containing extracted collocations (n-grams).

**-encoding value** input text file(s) character encoding

The value is the standard Java encoding (default encoding is UTF-8). If you have multiple input files, all of them should have the same character encoding.

**-tokenize** separate words on each line in input file

This option will break each word as a separate entry. The result will be simple list of words with a counter for each word. You should not use this option if you need to extract collocations.

**-gram value** maximum number of words in a phrase

This option value limits number of words in extracted phrases (collocations). Default number is 5, which means that longest collocation will contain 5 words separated by space.

**-count value** minimum times the phrase is repeated

This option removes all found collocations, which repeated less than specified value. Default number is 2, which means that all collocations repeated only once will be included.

**-lowercase** convert output to lowercase

This option converts input text to lowercase. It helps reducing variety of collocations, which represent the same combination of words, but in upper and lower case.

**-allowedwords filename** allowed word list to limit collocations

The value **filename** is a single file containing a list of words (one word per line), which are allowed to be present in collocations. This is very useful option to remove unwanted suggestions (for example, cleaning unwanted queries from the search query logs).

| | |
|---|---|
| **-verbose** | display progress information (on standard output) |

This option will print progress information on a screen, which can be redirected to a file for logging or used for debugging purpose.

## 5.2 Output file format

Output file represents the n-gram list, where each line contains a sequence of words separated by the space character. Each line has an integer number associated with the weight of this n-gram. The weight number is separated from words by tab character.

Example:

```
Serve as the incoming      92
Serve as the incubator     99
Serve as the independent   794
Serve as the index         223
Serve as the indication    72
Serve as the indicator     120
Serve as the indicators    45
```

## 5.3 Colocations Extractor usage examples

1. Create collocations from single text file search.log in UTF-8 character encoding and save output to a file collocations.txt:

```
java -mx1024m com.softcorporation.suggester.index.CollocationsExtractor -input search.log -output collocations.txt
```

2. Create collocations from two ISO Latin 1 word files, save output to a file collocations.txt:

```
java -mx1024m com.softcorporation.suggester.index.CollocationsExtractor -input “search 1.log,search 2.log” -output
collocations.txt -encoding ISO8859_1
```

3. Create collocations from all files located in directory /opt/suggester/logs/ with UTF-8 character encoding, save output to a file collocations.txt and print progress on the screen:

```
java –mx4096m com.softcorporation.suggester.index.CollocationsExtractor -input /opt/suggester/logs/ /opt/suggester/logs/
-output collocations.txt -encoding UTF-8 -verbose
```

## 5.4 Colocations Extractor limitations

The Collocations Extractor creates collocations in random access memory, which means the memory should be large enough to fit as much as possible n-grams in non-compressed form. More virtual memory your machine has - better will be the result. On average 2 GB of memory will be consumed to process about 200 Mb of text. You can, however, build collocations in stages, which allows overcame the memory limitation.

# 6. Suggester Configuration

The Suggester can be configured to fit your requirements. The configuration file represents plain text as name/value pairs and can be acessed as a standalone file or system resource in the classpath. Configuration file will be accessed as a system resource by default, which means that Suggester will try to use a classloader to find the configuration file. To access the configuration on the file system, you need to specify prefix file:// before the file name.

For example, to read configuration file “suggester.config” from directory: /opt/suggester/ use the following specification: file:///opt/suggester/suggester.config

## 6.1 Suggester Configuration

By default the file is located at the classpath: /com/softcorporation/suggester/basicSuggester.config

Parameters:
LENGTH_MIN_ED_1 - minimum word length to apply edit distance = 1.
LENGTH_MIN_ED_2 - minimum word length to apply edit distance = 2.
LENGTH_MIN_ED_3 - minimum word length to apply edit distance = 3.
LENGTH_MIN_ED_4 - minimum word length to apply edit distance = 4.
WEIGHT_EDIT_DISTANCE - edit distance weight for results sorting.
WEIGHT_SOUNDEX - soundex or metaphone weight for results sorting.
WEIGHT_LENGTH - word length difference weight. The weight for results sorting.
WEIGHT_LAST_CHAR - last character is different. The weight for results sorting.
WEIGHT_FIRST_CHAR - first character is different. The weight for results sorting.
WEIGHT_FIRST_CHAR_UPPER - first character is not in upper case. The weight for results sorting.
WEIGHT_FIRST_CHAR_LOWER - first character is not in lower case. The weight for results sorting.
WEIGHT_ADD_REM_CHAR - characters are added or removed. The weight for results sorting.
WEIGHT_FUZZY_PHON - Fuzzy matching. The weight for results sorting.
WEIGHT_JOINED_WORD - Joined word. The weight for results sorting.
SEARCH_JOINED - search for joined words.
REMOVE_JOINED_VARIATIONS - remove joined variations.
JOINED_WORD_LENGTH_MIN - minimum joined word length.
JOINED_WORD_LENGTH_EDT - minimum joined word length to consider edit distance = 1.
CLOSE_WORDS_CUT - remove unrelated suggestions.
DELIMITERS - word delimiters.
DELIMITERS_JOINED - joined words delimiters.

## 6.2 Language Configuration

The Fuzzy matching algorithm uses these files to select the best suggestion for the language. The file name should follow format: LANGUAGE.config. Creating your own language files you can add more languages to the Suggester.
The files are located at the classpath: /com/softcorporation/suggester/language/
Parameters:
LANGUAGE - the language identifier.
S1=S2:80[,Sn:##] - the relation (here it is 80) between strings S1 and S2, usually representing letters. The strongest relation = 100 (default). All language letters should be listed in the file, even if one letter has no relations to others.

# 7. Suggester Dictionary

The Suggester dictionary contains all words compiled in compressed format. The Advanced Suggester dictionary and Basic Suggester dictionary have different internal structure and are not compatible. Both dictionaries are additionally compressed using standard zip format.

The dictionary retains original word list. The dictionary internal structure supports UTF-8 encoding and keeps all original words in a case sensitive format.

The dictionary file will be accessed as a system resource by default, which means that Suggester will try to use a classloader to find the dictionary file. To access the dictionary on the file system, you need to specify prefix file:// before the file name.

For example, to read the dictionary file "dictionary.jar" from directory: /opt/suggester/ use the following specification: file:///opt/suggester/dictionary.jar

Dictionary limitations:

- The dictionary file name cannot be changed. If you need to modify the file name, you have to recompile the dictionary.
- The size of dictionary is limited by 2 GB of internal nodes.

## 8. Suggester Java API

The Suggester Java API allows you to interact with the Suggester in real time, to send request and receive a list of suggestions, as well load configuration and manipulate the index.

Main classes:

- AdvancedSuggester - searches multiple weighted dictionaries and returns suggestions.
- AdvancedDictionary – contains words or n-grams
- AdvancedSuggesterConfiguration – contains configuration parameters
- SuggesterException – the Exception with error message

The AdvancedSuggester class executes search for suggestions from one or multiple dictionaries. During initialization the AdvancedSuggester takes AdvancedSuggesterConfiguration class as a parameter.

Example of AdvancedSuggester Java usage:

```
String language = "en";
String dictFileName1 = "file://words_en-1.jar";
String dictFileName2 = "file://words_en-2.jar";
String configFileName = "file:///opt/suggester/advancedSuggester.config";

try
{
  // load dictionaries
  AdvancedDictionary dictionary1 = new AdvancedDictionary(dictFileName1);
  AdvancedDictionary dictionary2 = new AdvancedDictionary(dictFileName2);

  // load configuration
  AdvancedSuggesterConfiguration configuration = new
      AdvancedSuggesterConfiguration(configFileName);

  // attach dictionaries with weight
  AdvancedSuggester suggester = new AdvancedSuggester(configuration);

  // attach dictionaries with weight
  suggester.attach(dictionary1, 1);
  suggester.attach(dictionary2, 0.9);

  // search for up to 10 suggestions
  ArrayList suggestions = suggester.getSuggestions(word, 10, language);
  for (int i = 0; i < suggestions.size(); i++)
  {
    Suggestion suggestion = (Suggestion) suggestions.get(i);
    System.out.println(suggestion.getWord());
  }
}
catch (SuggesterException e)
{
  System.out.println("Error: " + e.getMessage());
}
```

More examples are provided in a package com.softcorporation.suggester.demo. See Suggester Java API documentation for details.