Urdu Component Development Project

Application Programming Interface (SpellCheck Utility)

August 17, 2007

CENTER FOR RESEARCH IN URDU LANGUAGE PROCESSING
NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES, LAHORE
PAKISTAN

Table of Contents

Revision History

Name	Change Date	Version	Description of Changes
Atif Gulzar	17-08-2007	1.0.0.0	Initial document

1 Introduction

SpellCheckerDll utility is an API that provides support for spelling check for Urdu language. It accepts a word and checks it for spelling errors. If an error is found, it generates a ranked list of suggested words. It is assumed that all the data used for the SpellCheckerDll utility is already in compact normalized form.

This document enlists SpellCheckerDll API with its complete specification.

In order to test the SpellCheckerDll API, a sample application has been developed. It is a Web application that uses SpellCheckerDll API and provides facility for spelling check for Urdu words. A word list of 149466 words is also provided with SpellCheckerDll API.

2 SpellCheckerDLL API

SpellCheckerDLL API is a dynamic link library that provides an interface for using SpellCheck utility functionalities. There are mainly two functions in the API:

- Initialize
- SpellCheck

2.1 *Initialize Function*

This function initializes the SpellCheckerDLL API. It must be called before invoking any other function of the SpellCheckerDLL API. This function reads the file names to be loaded/saved from config.ini file. If the initialize status is set to "YES" in config.ini file, the function loads the wordlist into hash table and generates the bigram and unigram files. Then it sets the initialize status to "NO". If the initialize status is set to "NO", the function loads the precompiled hash table, unigram and bigram.

Syntax

```
    bool Initialize ();
```

Return Value

If the function succeeds, it returns true, else false.

2.2 *SpellCheck Function*

Initialize function should be called before calling this function. This function takes a char* (a Unicode string) and checks it for spelling errors. If a spelling error is found, it returns a suggestedList containing the ranked list of suggested words.

Syntax

```
    int SpellCheck(unsigned char* spellerror, char **suggestedList);
```

Parameters

spellerror	Word to be tested for spelling errors.
suggestedList	Ranked list of at most 10 words if a spelling error is found. suggestlist must be freed after use.

Return Value

Returns "-1" if no spelling error is found, otherwise returns the size of suggestedlist (at most 10).

3 File Formats

3.1 *Config File*

This file contains the file paths to be used in the API. This file also contains initialization status of the application. If the initialize status is set to "YES" in config.ini, the *initialize* function loads the wordlist into hash table and generates the bigram and unigram files. Then it sets the initialize status to "NO". If the initialize status is set to "NO", the *initialize* function loads the precompiled hash table, unigram and bigram. This file should exist at the root path of application.

3.2 *Urdu Alphabet File*

This file (in Unicode format) contains a list of letters in Urdu alphabet, in order. Format of the file is simple; each line has only one letter. The order of letters is used to index into 'posting list index matrices or lists' (i.e. Bigram).

3.3 *Wordlist File*

This file (in Unicode format) contains a list of words with frequencies. The first line of this file contains the total number of words in the file. The remaining each line contains a word and its frequency.

3.4 *Bi-Gram Posting Lists File*

This file contains a number of posting lists. Each posting list contains indices to word forms in the hash table. These posting list indices are 4-byte numbers. A particular posting list contains indices to all words in which a particular bi-gram exists at any word position. These posting lists are arranged according to bi-grams and are referred from the offsets and counts in the Bi-Gram PL Index Matrix.

3.5 *Bi-Gram PL Index Matrix*

The 2D matrix contains indices to Bi-gram posting lists. A Bi-Gram is pair of characters that can occur in a word at any position. The matrix is a square matrix i.e. it has equal number of rows and columns, each of size equal to number of characters defined at start of the file. The matrix contains elements for all possible bi-gram combinations of allowed character set defined at start of file. Each matrix element (which is an index to one of the bi-gram posting lists) would take 6 bytes: 4 bytes for posting list offset and 2 bytes for number of following records in the posting list. Please note that, if a particular bi-gram does not exists i.e. if a particular element in the Bi-Gram PL Index Matrix does not point any word in the posting lists file, then its offset field would contain
-1 and count field would contain 0.

3.6 *Uni-Gram Posting Lists File*

The file contains a number of posting lists. Each posting list contains indices to word forms in the hash table. These posting list indices are 4-byte numbers. A particular posting list contains indices to all words having word length less than or equal to 3 characters, and in which a particular uni-gram exists at any word position. These posting lists are arranged according to uni-grams and are referred from the offsets and counts in the Uni-Gram PL Index List.

3.7 Uni-Gram PL Index List

Uni-Gram PL Index List is a list for uni-grams, which have word length less than or equal to 3 characters, and which can occur at any word position. Each list element contains offsets and count of posting list. If a particular uni-gram does not exist, its offset field would contain -1 and count field would contain 0. The size of the list is the same as the size of listed character set in Urdu alphabet file.

3.8 Hash Table File

This file is in binary format and contains all searchable lexemes (without diacritics). These word forms are given positions (or bucket) in table based of hashing. Universal hash function will be used to generate hash key. Linear chaining within the same array will be implemented for probing using “Next” field. Please note that entry can be single-word words or multi-words compounds. Each bucket is of fixed length and contains following fields:

Field	Word without Diacritics and spaces	Frequency	Next
Size (in Bytes)	40	4	4

Total Size (in Bytes) = 48

Detail of Lexicon Fields

Word without Diacritics

A null terminated string of 20 Unicode characters (total 40 bytes required) that represents word form lexeme without spaces and marked diacritics. Please note that all lexemes in the table may not be unique and duplication is allowed.

Frequency

A 4-byte integer that contains the frequency of a word obtained form a large corpus.

Next

A 4-byte integer that points to another bucket of the same Hash Table by storing the index of next bucket in the collision list. In result, a list of collided words is maintained in the same hash table. The last element of the list has 0 in its “Next” field. In case, a collision occurs, the collided word is added at the end of the list.