NAME

hunspell - spell checking, stemming, morphological generation and analysis

SYNOPSIS

#include <hunspell.hxx> /* or */
#include <hunspell.h>

Hunspell(const char ****affpath*, const char ****dpath***);**

Hunspell(const char ****affpath*, const char ****dpath***, const char * ***key***);**

~Hunspell()*;*

int add_dic(const char ****dpath*);**

int add_dic(const char ****dpath*, const char ****key***);**

int spell(const char ****word*);**

int spell(const char ****word*, int ****info***, char *******root***);**

int suggest(char****slst***, const char *****word*);

int analyze(char****slst***, const char *****word*);

int stem(char****slst***, const char *****word*);

int stem(char****slst***, char *****morph***, int** *n);*

int generate(char****slst***, const char *****word***, const char *****word2*);

int generate(char****slst***, const char *****word***, char *****desc***, int** *n);*

void free_list(char ******slst*, int** *n);*

**int add(const char *****word*);

int add_with_affix(const char *****word*, const char *****example*);

**int remove(const char *****word*);

**char * get_dic_encoding(***);*

**const char * get_wordchars(***);*

**unsigned short * get_wordchars_utf16(int ***len);*

**struct cs_info * get_csconv(***);*

**const char * get_version(***);*

DESCRIPTION

The **Hunspell** library routines give the user word-level linguistic functions: spell checking and correction, stemming, morphological generation and analysis in item-and-arrangement style.

The optional C header contains the C interface of the C++ library with Hunspell_create and Hunspell_destroy constructor and destructor, and an extra HunHandle parameter (the allocated object) in the wrapper functions (see in the C header file **hunspell.h**).

The basic spelling functions, **spell()** and **suggest()** can be used for stemming, morphological generation and analysis by XML input texts (see XML API).

Constructor and destructor

Hunspell's constructor needs paths of the affix and dictionary files. (In WIN32 environment, use UTF-8 encoded paths started with the long path prefix \\?\ to handle system-independent character encoding and very long path names, too.) See the **hunspell**(4) manual page for the dictionary format. Optional **key** parameter is for dictionaries encrypted by the **hzip** tool of the Hunspell distribution.

Extra dictionaries

The add_dic() function load an extra dictionary file. The extra dictionaries use the affix file of the allocated Hunspell object. Maximal number of the extra dictionaries is limited in the source code (20).

Spelling and correction

The spell() function returns non-zero, if the input word is recognised by the spell checker, and a zero value if not. Optional reference variables return a bit array (info) and the root word of the input word. Info bits checked with the SPELL_COMPOUND, SPELL_FORBIDDEN or SPELL_WARN macros sign compound words, explicit forbidden and probably bad words. From version 1.3, the non-zero return value is 2 for the dictionary words with the flag "WARN" (probably bad words).

The suggest() function has two input parameters, a reference variable of the output suggestion list, and an input word. The function returns the number of the suggestions. The reference variable will contain

the address of the newly allocated suggestion list or NULL, if the return value of suggest() is zero. Maximal number of the suggestions is limited in the source code.

The spell() and suggest() can recognize XML input, see the XML API section.

Morphological functions

The plain stem() and analyze() functions are similar to the suggest(), but instead of suggestions, return stems and results of the morphological analysis. The plain generate() waits a second word, too. This extra word and its affixation will be the model of the morphological generation of the requested forms of the first word.

The extended stem() and generate() use the results of a morphological analysis:

```
char ** result, result2;
int n1 = analyze(&result, "words");
int n2 = stem(&result2, result, n1);
```

The morphological annotation of the Hunspell library has fixed (two letter and a colon) field identifiers, see the **hunspell**(4) manual page.

```
char ** result;
char * affix = "is:plural"; // description depends from dictionaries, too
int n = generate(&result, "word", &affix, 1);
for (int i = 0; i < n; i++) printf("%s\n", result[i]);
```

Memory deallocation

The free_list() function frees the memory allocated by suggest(), analyze, generate and stem() functions.

Other functions

The add(), add_with_affix() and remove() are helper functions of a personal dictionary implementation to add and remove words from the base dictionary in run-time. The add_with_affix() uses a second root word as the model of the enabled affixation and compounding of the new word.

The get_dic_encoding() function returns "ISO8859-1" or the character encoding defined in the affix file with the "SET" keyword.

The get_csconv() function returns the 8-bit character case table of the encoding of the dictionary.

The get_wordchars() and get_wordchars_utf16() return the extra word characters defined in affix file

for tokenization by the "WORDCHARS" keyword.

The get_version() returns the version string of the library.

XML API

The spell() function returns non-zero for the "<?xml?>" input indicating the XML API support.

The suggest() function stems, analyzes and generates the forms of the input word, if it was added by one of the following "SPELLML" syntaxes:

```
<?xml?>
<query type="analyze">
<word>dogs</word>
</query>

<?xml?>
<query type="stem">
<word>dogs</word>
</query>

<?xml?>
<query type="generate">
<word>dog</word>
<word>cats</word>
</query>

<?xml?>
<query type="generate">
<word>dog</word>
<code><a>is:pl</a><a>is:poss</a></code>
</query>

<?xml?>
<query type="add">
<word>word</word>
</query>

<?xml?>
<query type="add">
<word>word</word>
```

<word>model_word_for_affixation_and_compounding</word>
</query>

The outputs of the type="stem" query and the stem() library function are the same. The output of the type="analyze" query is a string contained a <code><a>result1<a>result2...</code> element. This element can be used in the second syntax of the type="generate" query.

EXAMPLE

See analyze.cxx in the Hunspell distribution.

AUTHORS

Hunspell based on Ispell's spell checking algorithms and OpenOffice.org's Myspell source code.

Author of International Ispell is Geoff Kuenning.

Author of MySpell is Kevin Hendricks.

Author of Hunspell is Làszlò Nèmeth.

Author of the original C API is Caolan McNamara.

Author of the Aspell table-driven phonetic transcription algorithm and code is Björn Jacke.

See also THANKS and Changelog files of Hunspell distribution.