LibTehthu

0.2

Generated by Doxygen 1.6.3

Wed Aug 4 10:04:16 2010

Contents

Chapter 1

Namespace Index

1.1 Package List

Here are the packages with brief descriptions (if available):

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Namespace Documentation

4.1 Package LibTehthu

The main LibTehthu translator namespace.

Classes

- class StringCase

  The StringCase class. A class for detecting the case of, and transforming the case of, letters in a string.
  A "letter" is a special type of character that is defined as having both a capital and a lowercase type.
  A "symbol" is a character that does not have any such capital/lowercase modality.
  Capital and lowercase letters have different character codes. The mapping between capital and lower letters is defined by the CLR.
  Specifically, the routines Char.ToUpper() and Char.ToLower() are used.

- class Tehthu

  Main Tehthu class.

Enumerations

- enum CaseType {

  Caps, **Proper**, **Lower**, **Mixed**,

  Null }

  The types of caseness for a word. Caps: Every letter in the word is a capital letter.
  Proper: Only the first letter of the word is a capital letter; the rest are lowercase.
  Lower: Every letter in the word is a lowercase letter.
  Mixed: None of the above cases (Caps, Proper, Lower) apply, but the word contains at least one letter.
  Null: The word contains no letters (only symbols), is null, or zero-length.

- enum TranslateDirection { **LeftToRight**, **RightToLeft** }

  Enum for specifying which direction to translate.

4.1.1 Detailed Description

The main LibTehthu translator namespace. You should be sure to add to your code a "using" line for this namespace, to avoid having to type it repeatedly.

4.1.2 Enumeration Type Documentation

4.1.2.1 enum LibTehthu::TranslateDirection

Enum for specifying which direction to translate.

LeftToRight means that the language on the left-hand side of the delimiter will be translated to the language on the right-hand side.

RightToLeft means the opposite.

To query the names of the detected languages in the current dictionary, call getLeftLanguageName() or getRightLanguageName().

Chapter 5

Class Documentation

5.1 SizeQueue< T > Class Template Reference

Public Member Functions

- **SizeQueue** (int maxSize)
- void **Enqueue** (T item)
- T **Dequeue** ()

template<T> class SizeQueue< T >

The documentation for this class was generated from the following file:

- SizeQueue.cs

5.2 LibTehthu.StringCase Class Reference

The StringCase class. A class for detecting the case of, and transforming the case of, letters in a string.

A "letter" is a special type of character that is defined as having both a capital and a lowercase type.

A "symbol" is a character that does not have any such capital/lowercase modality.

Capital and lowercase letters have different character codes. The mapping between capital and lower letters is defined by the CLR.

Specifically, the routines Char.ToUpper() and Char.ToLower() are used.

Static Public Member Functions

- static string transformCase (string s, CaseType trans)

  Convert the caseness of a string into the specified type.

- static CaseType getCaseType (string s)

  Determine the case type of the specified string.

5.2.1 Detailed Description

The StringCase class. A class for detecting the case of, and transforming the case of, letters in a string.

A "letter" is a special type of character that is defined as having both a capital and a lowercase type.

A "symbol" is a character that does not have any such capital/lowercase modality.

Capital and lowercase letters have different character codes. The mapping between capital and lower letters is defined by the CLR.

Specifically, the routines Char.ToUpper() and Char.ToLower() are used.

5.2.2 Member Function Documentation

5.2.2.1 static CaseType LibTehthu.StringCase.getCaseType (string *s*) `[static]`

Determine the case type of the specified string.

Returns

The case type.

See the documentation of the CaseType enum for details.

5.2.2.2 static string LibTehthu.StringCase.transformCase (string *s*, CaseType *trans*) `[static]`

Convert the caseness of a string into the specified type.

Parameters

s The string to convert.

trans The transformation to perform.

Returns

The transformed string.

Symbols (whitespace, non-letters) are unmodified. Letter characters are modified as dictated by the CaseType:

Caps: Every letter in the string is converted to a capital letter.

Proper: The first letter of the word is made capital; the rest are made lowercase.

Lower: Every letter in the string is converted to a lowercase letter.

Mixed: The string is returned unmodified, since there are many ways to create a mixed case string, and we don't want to choose one arbitrarily.

Null: A null reference is returned.

The documentation for this class was generated from the following file:

- StringCase.cs

5.3 LibTehthu.Tehthu Class Reference

Main Tehthu class.

Public Member Functions

- Tehthu (string path)

  Instantiates the translator.

- Tehthu (FileInfo fileinfo)

  Instantiates the translator.

- Tehthu (string path, string delim)

  Instantiates the translator.

- Tehthu (FileInfo fileinfo, string delim)

  Instantiates the translator.

- Tehthu (FileInfo fileinfo, string delim, string left, string right)

  Instantiates the translator.

- void setDelimiter (string delim)

  Set the delimiter that separates the left hand language lines from the right hand. WRITE (this function is not thread-safe).
  This function forces a reparse of the dictionary.

- string getDelimiter ()

  Retrieve the delimiter that separates the left hand language from the right hand in the dictionary file.
  .

- void setFile (string path)

  Set the dictionary file used to translate.

- void setFile (FileInfo fileinfo)

  Set the dictionary file used to translate.

- FileInfo getFile ()

  Get the dictionary file used to translate.

- void setLeftLanguageName (string name)

  Set the user-friendly language name of the left-hand language.

- void setRightLanguageName (string name)

  Set the user-friendly language name of the right-hand language.

- string getLeftLanguageName ()

  Get the user-friendly language name of the left-hand language.

- string getRightLanguageName ()

  Get the user-friendly language name of the right-hand language.

- string translate (string input, TranslateDirection dir)

  Translate an input sentence into an output sentence.

- bool translateWord (string _origKey, out string val, TranslateDirection dir)

  Translate a single word from an input language to an output language.

- string takeLogLine ()

  Wait until the next line of text is available in the log, then retrieve it.

- void putLogLine (string item)

  Add a line to the log buffer.

- void connectToLog ()

  Notify LibTehthu that the application intends to handle the log buffer.

- void disconnectFromLog ()

  Notify LibTehthu that the application does NOT intend to handle the log buffer. This is the default.

- bool haveLogConnection ()

  Indicate whether the log is currently being filled with new messages.

- void reparse ()

  (re)parse the dictionary file, discarding the in-memory database and storing a fresh copy from disk.

5.3.1 Detailed Description

Main Tehthu class. You must instantiate an instance of this class to use the translator.

Multiple instances are supported because there is no static data.

Tehthu is not thread-safe; however, it is possible to use read-only functions from multiple threads.

It is important to make sure that only one thread at a time is accessing any of the functions marked as "WRITE" in this documentation.

That is, only one WRITE function may be executing at a time, and none of the WRITE functions are reentrant.

If you use multiple threads, you should make sure all readers wait whenever one thread uses a WRITE function.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 LibTehthu.Tehthu.Tehthu (string *path*)

Instantiates the translator.

Create a new translator with the default delimiter and default language names from the given dictionary file.

The language names in the dictionary (if any) will not be set until you call reparse().

5.3.2.2 LibTehthu.Tehthu.Tehthu (FileInfo *fileinfo*)

Instantiates the translator.

Create a new translator with the default delimiter and default language names from the given dictionary file.

The language names in the dictionary (if any) will not be set until you call reparse().

5.3.2.3 LibTehthu.Tehthu.Tehthu (string *path*, string *delim*)

Instantiates the translator.

Create a new translator with the given delimiter and default language names from the given dictionary file.

The language names in the dictionary (if any) will not be set until you call reparse().

5.3.2.4 LibTehthu.Tehthu.Tehthu (FileInfo *fileinfo*, string *delim*)

Instantiates the translator.

Create a new translator with the given delimiter and default language names from the given dictionary file.

The language names in the dictionary (if any) will not be set until you call reparse().

5.3.2.5 LibTehthu.Tehthu.Tehthu (FileInfo *fileinfo*, string *delim*, string *left*, string *right*)

Instantiates the translator.

Create a new translator with the given delimiter and language names from the given dictionary file.

The language names in the dictionary (if any) will override the language names set here when you call reparse().

5.3.3 Member Function Documentation

5.3.3.1 void LibTehthu.Tehthu.connectToLog ()

Notify LibTehthu that the application intends to handle the log buffer.

Unless this function or takeLogLine() is called, the log buffer will not be filled.

WRITE (this function is not thread-safe).

5.3.3.2 void LibTehthu.Tehthu.disconnectFromLog ()

Notify LibTehthu that the application does NOT intend to handle the log buffer. This is the default.

Once this function is called, the log buffer will no longer be filled with new log lines.

If the log buffer is not empty when called, this function will not delete its contents.

WRITE (this function is not thread-safe).

5.3.3.3 string LibTehthu.Tehthu.getDelimiter ()

Retrieve the delimiter that separates the left hand language from the right hand in the dictionary file.

.

Returns

The delimiter, usually one character. The default is "|" (no quotes).

5.3.3.4 FileInfo LibTehthu.Tehthu.getFile ()

Get the dictionary file used to translate.

Returns

A FileInfo object containing information about the file being used as the dictionary.

5.3.3.5 bool LibTehthu.Tehthu.haveLogConnection ()

Indicate whether the log is currently being filled with new messages.

Returns

true if connectToLog() or takeLogLine() has been called more recently than disconnectFromLog(); false otherwise.

5.3.3.6 void LibTehthu.Tehthu.putLogLine (string *item*)

Add a line to the log buffer.

This function will put a line in the log's blocking queue, potentially blocking if it is full.

The limit on the number of lines that can be logged without having any lines removed with takeLogLine() is currently hard-coded at 100. Additionally, this function is a no-op if neither takeLogLine() nor connectToLog() has been called for this Tehthu instance. This prevents the buffer from filling up if the application is not trying to handle the log buffer.

This function is thread-safe.

5.3.3.7 void LibTehthu.Tehthu.reparse ()

(re)parse the dictionary file, discarding the in-memory database and storing a fresh copy from disk.

This function should be called if the dictionary is known to have been modified.

It is also called internally by LibTehthu (as indicated in this documentation) when certain WRITE functions are called.

WRITE (this function is not thread-safe).

5.3.3.8 void LibTehthu.Tehthu.setFile (FileInfo *fileinfo*)

Set the dictionary file used to translate.

WRITE (this function is not thread-safe).

This function forces a reparse of the dictionary.

5.3.3.9 void LibTehthu.Tehthu.setFile (string *path*)

Set the dictionary file used to translate.

WRITE (this function is not thread-safe).

This function forces a reparse of the dictionary.

5.3.3.10 void LibTehthu.Tehthu.setLeftLanguageName (string *name*)

Set the user-friendly language name of the left-hand language.

WRITE (this function is not thread-safe).

5.3.3.11 void LibTehthu.Tehthu.setRightLanguageName (string *name*)

Set the user-friendly language name of the right-hand language.

WRITE (this function is not thread-safe).

5.3.3.12 string LibTehthu.Tehthu.takeLogLine ()

Wait until the next line of text is available in the log, then retrieve it.

Returns

Returns one line of text, conventionally without any newline characters.

This function will block the current thread until a line of text is available in the log.

If the log is currently disconnected, this function will connect it.

Lines are written to the log when there are non-fatal errors in the translation or parsing stages.

Each line put in the buffer is returned in a takeLogLine() call exactly once.

This function is intended to be used in a loop in a separate thread from the main thread using Tehthu.

This function is thread-safe if connectToLog() has already been called, otherwise it is not.

This function does not schedule multiple readers fairly; they are scheduled arbitrarily by the CLR.

Example ThreadStart func:

```
void threadStart()
{
while(true)
{
try
```

```
{
do_something_with(tehthu.takeLogLine());
}
catch(ThreadAbortException tae)
{
break;
}
}
}
```

5.3.3.13 string LibTehthu.Tehthu.translate (string *input*, TranslateDirection *dir*)

Translate an input sentence into an output sentence.

Parameters

input The input sentence.

dir The translation direction. LeftToRight means the input sentence is in the left-hand language, and the output sentence is in the right-hand language. Vice-versa for RightToLeft.

Returns

The translated sentence in the output language.

A "sentence" is a string of one or more words, possibly containing symbols.

A word is a substring separated from the rest of the string by whitespace.

Sentences MAY contain the delimiter character used in the dictionary, but it will not be translated.

Sentences MAY contain newlines, but it is preferred that each sentence does not contain any newlines.

Any newlines present in the input sentence will be converted to spaces.

Words that are not translated are omitted from the output sentence.

This function is reentrant, and can be used from multiple threads as long as WRITE functions are synchronized.

5.3.3.14 bool LibTehthu.Tehthu.translateWord (string *_origKey*, out string *val*, TranslateDirection *dir*)

Translate a single word from an input language to an output language.

Parameters

_origKey The input word. Symbols are stripped from the beginning and end of the word before it is translated.

val An "out" parameter for the output word. It will not be set if the function returns false.

dir The translation direction. LeftToRight means the input sentence is in the left-hand language, and the output sentence is in the right-hand language. Vice-versa for RightToLeft.

Returns

true if the word was translated; false if no translation was found in the dictionary.

No attempt is made to tokenize the string, so the entire input string will be looked up in the dictionary. Therefore you should call translate() instead if you want to translate a string containing multiple tokens. This function is reentrant, and can be used from multiple threads as long as WRITE functions are synchronized.

The documentation for this class was generated from the following file:

- Tehthu.cs

Chapter 6

File Documentation

6.1 StringCase.cs File Reference

Utilities for dealing with capital and lowercase letters in creative ways.

Classes

- class LibTehthu.StringCase

  The StringCase class. A class for detecting the case of, and transforming the case of, letters in a string.
  A "letter" is a special type of character that is defined as having both a capital and a lowercase type.
  A "symbol" is a character that does not have any such capital/lowercase modality.
  Capital and lowercase letters have different character codes. The mapping between capital and lower letters is defined by the CLR.
  Specifically, the routines Char.ToUpper() and Char.ToLower() are used.

Packages

- package LibTehthu

  The main LibTehthu translator namespace.

Enumerations

- enum LibTehthu.CaseType {

  Caps, **Proper**, **Lower**, **Mixed**,

  Null }

  The types of caseness for a word. Caps: Every letter in the word is a capital letter.
  Proper: Only the first letter of the word is a capital letter; the rest are lowercase.
  Lower: Every letter in the word is a lowercase letter.
  Mixed: None of the above cases (Caps, Proper, Lower) apply, but the word contains at least one letter.
  Null: The word contains no letters (only symbols), is null, or zero-length.

6.1.1 Detailed Description

Utilities for dealing with capital and lowercase letters in creative ways. Tehthu.cs is one-way tightly coupled to the StringCase API.

6.2 Tehthu.cs File Reference

The main LibTehthu translator.

Classes

- class LibTehthu.Tehthu

  Main Tehthu class.

Packages

- package LibTehthu

  The main LibTehthu translator namespace.

Enumerations

- enum LibTehthu.TranslateDirection { **LeftToRight**, **RightToLeft** }

  Enum for specifying which direction to translate.

6.2.1 Detailed Description

The main LibTehthu translator. LibTehthu performs three primary functions, usually in this order:

1. Parses a Tehthu dictionary file, which contains word translations from one language to another.

2. Tokenizes input sentences by dividing the sentence into words. Words can not contain spaces because each space denotes a new word.

3. Translates the input words into the output words, then re-assembles the sentence with the translated words in the same order and in the same case.

Terminology and Example File:

The "left-hand language" is the language to the left of the delimiter in the dictionary file.

Correspondingly, the right-hand language is the language to the right of the delimiter.

Here is a simple example of a one-word dictionary:

[English] [Spanish]

hello|hola

In this example, English is the left-hand language name, Spanish is the right-hand language name.

'hello' is a word in the left-hand language, and 'hola' is a word in the right-hand language.

DETAILED FILE FORMAT

The program does not care about the extension of the file. However, the de facto standard for the extension is .teh, after Tehthu.

I. Language Names

Description: A dictionary file may optionally start out with a stanza matching the following regular expression:

\[.+\] \[.+\]

This will make the core LibTehthu aware of user-friendly names to refer to the languages used in the dictionary. These are also exposed in the Gtk-GUI.

Example: [Foo] [Bar]

II. Mappings

Description: A mapping is a delimiter-separated pair of values matching the following regular expression:

.+<delim>.+

where <delim> is replaced with a string that is never used in the construction of the tokens on either side. The mappings are the core of the dictionary.

Example: hello|hola

Rules:

(a) There can be a maximum of one Language Names stanza and an unlimited number of Mappings stanzas (limited by system resources).

(b) The Language Names stanza MUST be the first line in the file, if it is included. However, it is optional whether to include it at all.

(c) Each stanza is separated by any new line pattern recognized by the .NET CLR.

Currently that means you can set your line endings to

(the default on UNIX) or

(the default on Windows).

III. Character encodings

Character encodings that can be automatically detected by the System.IO utilities of .NET are supported.

For example, ASCII, UTF-8 and Windows Western are supported. Multi-byte encodings, variable-length encodings, and non-Western characters are COMPLETELY UNTESTED.