PDF Search Engine

Generated by Doxygen 1.8.11

Contents

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Chapter 2

Namespace Index

2.1 Namespace List

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

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

Module Documentation

5.1 Indexing

Library for stemming words down to their root words.

5.1.1 Detailed Description

Library for stemming words down to their root words.

Date

2003-2016

Copyright

Oleander Software, Ltd.

Author

Oleander Software, Ltd.

5.2 Debugging

Functions used for debugging.

Macros

- #define **CASSERT**(x) typedef char __C_ASSERT__[(x) ? 1 : -1]
- #define **NON_UNIT_TEST_ASSERT**(x) assert(x)
- #define **DUMP_TO_FILE**(x, file) __debug::__dump_to_file((x), (file))

5.2.1 Detailed Description

Functions used for debugging.

Date

2008-2015

Copyright

Oleander Software, Ltd.

Author

Oleander Software, Ltd.

This program is free software; you can redistribute it and/or modify it under the terms of the BSD License.

5.2.2 Macro Definition Documentation

5.2.2.1 #define CASSERT(*x*) typedef char __C_ASSERT__[(x) ? 1 : -1]

Validates that an expression is true at compile time. If the expression is false then compilation will fail.

5.2.2.2 #define DUMP_TO_FILE(*x, file*) __debug::__dump_to_file((x), (file))

Prints data stream to a specified file.

5.2.2.3 #define NON_UNIT_TEST_ASSERT(*x*) assert(x)

If unit test symbol (__UNITTEST) is defined then does nothing; otherwise asserts. This is useful for suppressing asserts when unit testing.

5.3 Stemming

Library for stemming words down to their root words.

Namespaces

- **stemming**
  Namespace for stemming classes.

5.3.1 Detailed Description

Library for stemming words down to their root words.

Date

2003-2015

Copyright

Oleander Software, Ltd.

Author

Oleander Software, Ltd.

This program is free software; you can redistribute it and/or modify it under the terms of the BSD License.

5.4 Mathematics

Math and statistics classes.

Classes

- class **double_less**

  "less" interface for double values.

Functions

- template<typename T >
  T **safe_modulus** (const T dividend, const T divisor)

  Modulus operation that checks for modulus by zero or into zero (returns zero for those situations).
- template<typename T >
  T **safe_divide** (const T dividend, const T divisor)

  Division operation that checks for division by zero or into zero (returns zero for those situations).
- bool **compare_doubles** (const double actual, const double expected, const double delta=1e-6)

  Compares two double values (given the specified precision).
- bool **compare_doubles_less** (const double left, const double right, const double delta=1e-6)

  Compares two double values for less than (given the specified precision).
- bool **compare_doubles_less_or_equal** (const double left, const double right, const double delta=1e-6)

  Compares two double values for less than or equal to (given the specified precision).
- bool **compare_doubles_greater** (const double left, const double right, const double delta=1e-6)

  Compares two double values for greater than (given the specified precision).
- bool **double_less::operator()** (const double &left, const double &right) const
- template<typename T >
  bool **int_to_bool** (const T intVal)

  Converts an integral type to a boolean. Compilers complain about directly assigning an int to a bool (casting doesn't help either), so this works around that.

5.4.1 Detailed Description

Math and statistics classes.

Date

2004-2015

Copyright

Oleander Software, Ltd.

Author

Oleander Software, Ltd.

This program is free software; you can redistribute it and/or modify it under the terms of the BSD License.

5.4.2 Function Documentation

5.4.2.1 bool compare_doubles (const double *actual*, const double *expected*, const double *delta* = `1e-6`) `[inline]`

Compares two double values (given the specified precision).

Parameters

actual	The value being reviewed.
expected	The expected value to compare against.
delta	The tolerance of how different the values can be. The larger the delta, the higher precision used in the comparison.

Returns

True if the value matches the expected value.

5.4.2.2 bool compare_doubles_greater (const double *left,* const double *right,* const double *delta =* `1e-6`) `[inline]`

Compares two double values for greater than (given the specified precision).

Parameters

left	The value being reviewed.
right	The other value to compare against.
delta	The tolerance of how different the values can be. The larger the delta, the higher precision used in the comparison.

Returns

True if the value is greater than the other value.

5.4.2.3 bool compare_doubles_less (const double *left,* const double *right,* const double *delta =* `1e-6`) `[inline]`

Compares two double values for less than (given the specified precision).

Parameters

left	The value being reviewed.
right	The other value to compare against.
delta	The tolerance of how different the values can be. The larger the delta, the higher precision used in the comparison.

Returns

True if the value is less than the other value.

5.4.2.4 bool compare_doubles_less_or_equal (const double *left,* const double *right,* const double *delta =* `1e-6`) `[inline]`

Compares two double values for less than or equal to (given the specified precision).

Parameters

left	The value being reviewed.
right	The other value to compare against.
delta	The tolerance of how different the values can be. The larger the delta, the higher precision used in the comparison.

Returns

True if the value is less than or equal to the other value.

5.4.2.5 template<typename T > bool int_to_bool (const T *intVal*) `[inline]`

Converts an integral type to a boolean. Compilers complain about directly assigning an int to a bool (casting doesn't help either), so this works around that.

Parameters

intVal	The integer value to convert to a boolean.

Returns

The boolean equivalent of the integer.

5.4.2.6 template<typename T > T safe_divide (const T *dividend,* const T *divisor*) `[inline]`

Division operation that checks for division by zero or into zero (returns zero for those situations).

Parameters

dividend	The dividend (i.e., the value being divided).
divisor	The divisor (i.e., the value dividing by).

Returns

The quotient of the division operation, or zero if one of the values was invalid.

Note

If the template type has floating point precision, then the result will retain its precision.

5.4.2.7 template<typename T > T safe_modulus (const T *dividend,* const T *divisor*) `[inline]`

Modulus operation that checks for modulus by zero or into zero (returns zero for those situations).

Parameters

dividend	The dividend (i.e., the value being divided).
divisor	The divisor (i.e., the value dividing by).

Returns

The remainder of the modulus operation, or zero if one of the values was invalid.

5.5 StringOperations

Classes

- class **string_util::string_tokenize**$<$ **T** $>$

  Tokenizes a string using a set of delimiters.

5.5.1 Detailed Description

Classes for string operations.

5.6 Utilities

Utility classes.

Classes

- class **within**< **T** >
  Determines if a value is within a given range.
- class **comparable_first_pair**< **T1, T2** >
  pair interface that compares on the first item
- class **backup_variable**< **T** >
  class that remembers its original value from construction.

Macros

- #define **size_of_array**(x) (sizeof(x)/sizeof(x[0]))

Functions

- template<typename T >
  T **within_range** (const T start, const T end, const T value)
- template<typename T >
  bool **is_within** (const T value, const T first, const T second)
- **within**< **T** >**::within** (T range_begin, T range_end)
- bool **within**< **T** >**::operator()** (T value) const
- **comparable_first_pair**< **T1, T2** >**::comparable_first_pair** (const T1 &t1, const T2 &t2)
- bool **comparable_first_pair**< **T1, T2** >**::operator**< (const **comparable_first_pair**< T1, T2 > &that) const
- bool **comparable_first_pair**< **T1, T2** >**::operator==** (const **comparable_first_pair**< T1, T2 > &that) const
- **backup_variable**< **T** >**::backup_variable** (const T &value)
- void **backup_variable**< **T** >**::operator=** (const T &value)
- bool **backup_variable**< **T** >**::operator==** (const T &value) const
- bool **backup_variable**< **T** >**::operator**< (const T &value) const
- bool **backup_variable**< **T** >**::operator<=** (const T &value) const
- bool **backup_variable**< **T** >**::operator**> (const T &value) const
- bool **backup_variable**< **T** >**::operator>=** (const T &value) const
- void **backup_variable**< **T** >**::operator+** (const T &value)
- void **backup_variable**< **T** >**::operator+=** (const T &value)
- void **backup_variable**< **T** >**::operator-** (const T &value)
- void **backup_variable**< **T** >**::operator-=** (const T &value)
- **backup_variable**< **T** >**::operator const T** () const
- T ∗ **backup_variable**< **T** >**::operator&** ()
- const T & **backup_variable**< **T** >**::get_value** () const
- T & **backup_variable**< **T** >**::get_value** ()
- bool **backup_variable**< **T** >**::has_changed** () const
- template<typename T >
  bool **is_either** (const T value, const T first, const T second)
  Determines if a given value is either of two other given values.
- template<typename T >
  bool **is_neither** (const T value, const T first, const T second)

Determines if a given value is neither of two other given values.

- template< typename inT , typename outT , typename member_extract_functorT >
  outT **copy_member** (inT begin, inT end, outT dest, member_extract_functorT get_value)
- template< typename inT , typename outT , typename _Pr , typename member_extract_functorT >
  outT **copy_member_if** (inT begin, inT end, outT dest, _Pr meets_criteria, member_extract_functorT get_↩ value)

  Copies a member value between objects based on specified criteria.

5.6.1 Detailed Description

Utility classes.

Date

2003-2015

Copyright

Oleander Software, Ltd.

Author

Oleander Software, Ltd.

This program is free software; you can redistribute it and/or modify it under the terms of the BSD License.

5.6.2 Macro Definition Documentation

5.6.2.1 #define size_of_array(*x*) (sizeof(x)/sizeof(x[0]))

Returns

The item count of an array.

Note

Do not call this on an empty array. Also, this is meant for arrays of intrinsic types only.

5.6.3 Function Documentation

5.6.3.1 template< typename inT , typename outT , typename member_extract_functorT > outT copy_member (inT *begin,* inT *end,* outT *dest,* member_extract_functorT *get_value*) `[inline]`

calls a member function of elements in a container for each element in another container

5.6.3.2 template< typename T > bool is_within (const T *value,* const T *first,* const T *second*) `[inline]`

Returns

True if a value is within a given range.

5.6.3.3 template< typename T > bool within< T >::operator() (T *value*) const `[inline]`

Returns

True if value is within the valid range of values.

Parameters

value	The value to review.

5.6.3.4 template<typename T > within< T >::within (T *range_begin,* T *range_end*) [inline]

Constructor.

Parameters

range_begin	The beginning of the valid range.
range_end	The end of the valid range.

5.6.3.5 template<typename T > T within_range (const T *start,* const T *end,* const T *value*) [inline]

Range checks a given value and truncates it if it is too high or low.

Parameters

start	The start of the valid range.
end	The end of the valid range.
value	The value to be range checked.

Returns

The value if within the valid range. If it was too large, then the end of the range is returned. If too low, then the start of the range is returned.

Chapter 6

Namespace Documentation

6.1 stemming Namespace Reference

Namespace for stemming classes.

Classes

- class **english_stem**

  English stemmer.
- class **no_op_stem**
- class **stem**

  The base class for language-specific stemmers.

Enumerations

- enum **stemming_type** {
  no_stemming, **danish**, **dutch**, **english**,
  finnish, **french**, **german**, **italian**,
  norwegian, **portuguese**, **spanish**, **swedish**,
  STEMMING_TYPE_COUNT }

Variables

- const wchar_t **UPPER_Y_HASH** = 7
- const wchar_t **LOWER_Y_HASH** = 9
- const wchar_t **UPPER_I_HASH** = 10
- const wchar_t **LOWER_I_HASH** = 11
- const wchar_t **UPPER_U_HASH** = 12
- const wchar_t **LOWER_U_HASH** = 13

6.1.1 Detailed Description

Namespace for stemming classes.

6.2 string_util Namespace Reference

Classes

- class **equal_basic_string_i_compare**
- class **equal_string_compare**
- class **equal_string_i_compare**
- class **less_basic_string_compare**
- class **less_string_compare**
- class **less_string_i_compare**
- class **less_string_n_compare**
- class **less_string_natural_order_i_compare**
- class **less_string_ni_compare**
- class **string_tokenize**

  Tokenizes a string using a set of delimiters.
- class **string_trim**

  trims whitespace from around a string

Functions

- wchar_t **tolower_western** (const wchar_t c)

  lowercases any Western European alphabetic characters
- double **strtol** (const char ∗str, char ∗∗strend, int radix)
- double **strtol** (const wchar_t ∗str, wchar_t ∗∗strend, int radix)
- double **strtod** (const char ∗str, char ∗∗strend)

  strtod
- double **strtod** (const wchar_t ∗str, wchar_t ∗∗strend)
- int **atoi** (const char ∗str)

  atoi
- int **atoi** (const wchar_t ∗str)
- long **atol** (const char ∗str)

  atol
- long **atol** (const wchar_t ∗str)
- int **tolower** (char c)

  tolower
- wchar_t **tolower** (wchar_t c)
- int **toupper** (char c)

  toupper
- wchar_t **toupper** (wchar_t c)
- template<typename T >
  T ∗ **memset** (T ∗dest, int c, size_t count)

  memset
- char ∗ **memset** (char ∗dest, int c, size_t count)
- wchar_t ∗ **memset** (wchar_t ∗dest, int c, size_t count)
- const char ∗ **strchr** (const char ∗s, int ch)

  strchr
- const wchar_t ∗ **strchr** (const wchar_t ∗s, wchar_t ch)
- const char ∗ **strstr** (const char ∗s1, const char ∗s2)

  strstr
- const wchar_t ∗ **strstr** (const wchar_t ∗s1, const wchar_t ∗s2)
- size_t **strcspn** (const char ∗string1, const char ∗string2)

strcspn
- size_t **strcspn** (const wchar_t ∗string1, const wchar_t ∗string2)
- char ∗ **strncat** (char ∗strDest, const char ∗strSource, size_t count)

  strncat
- wchar_t ∗ **strncat** (wchar_t ∗strDest, const wchar_t ∗strSource, size_t count)
- int **wctomb** (wchar_t ∗s, wchar_t wc)

  wctomb
- int **wctomb** (char ∗s, wchar_t wc)
- size_t **strlen** (const char ∗text)
- size_t **strlen** (const wchar_t ∗text)
- int **strcmp** (const char ∗string1, const char ∗string2)

  strcmp
- int **strcmp** (const wchar_t ∗string1, const wchar_t ∗string2)
- int **strncmp** (const char ∗string1, const char ∗string2, size_t count)

  strncmp
- int **strncmp** (const wchar_t ∗string1, const wchar_t ∗string2, size_t count)
- char ∗ **strncpy** (char ∗strDest, const char ∗strSource, size_t count)

  strncpy
- wchar_t ∗ **strncpy** (wchar_t ∗strDest, const wchar_t ∗strSource, size_t count)
- template<typename charT >
  int **itoa** (long value, charT ∗out, const size_t length)

  functions not available in ANSI C
- template<typename T >
  bool **is_space** (const T ch)
- template<typename T >
  bool **is_hex_digit** (const T ch)
- template<typename T >
  int **axtoi** (const T ∗hexStr, size_t length=-1)
- template<typename T >
  size_t **strnlen** (const T ∗str, const size_t maxlen)
- template<typename T >
  const T ∗ **stristr** (const T ∗string, const T ∗strSearch)

  search for substring in string (case-insensitive)
- template<typename T >
  const T ∗ **strnistr** (const T ∗string, const T ∗strSearch, const size_t string_len)
- template<typename T >
  const T ∗ **strrstr** (const T ∗string, const T ∗search, size_t offset)
- template<typename T >
  int **strnicmp** (const T ∗first, const T ∗last, size_t count)

  Case-insensitive comparison by character count.
- template<typename T >
  int **stricmp** (const T ∗first, const T ∗last)

  Case-insensitive comparison.
- template<typename T >
  int **strnatordcmp** (const T ∗first_string, const T ∗second_string, bool case_insensitive=false)
- template<typename T >
  int **strnatordncasecmp** (const T ∗a, const T ∗b)

  Compare, recognizing numeric strings and ignoring case.
- template<typename T >
  bool **has_suffix** (const T ∗text, const size_t text_length, const T ∗suffix, const size_t suffix_length)
- template<typename T >
  const T ∗ **find_matching_close_tag** (const T ∗string, const T openSymbol, const T closeSymbol, const bool fail_on_overlapping_open_symbol=false)

- template<typename T >
  const T ∗ **find_matching_close_tag** (const T ∗string, const T ∗openSymbol, const T ∗closeSymbol)

  Searches for a matching tag, skipping an extra open/close pairs of symbols in between.
- template<typename T >
  const T ∗ **strnchr** (const T ∗string, const T ch, size_t numberOfCharacters)
- template<typename T >
  const T ∗ **strcspn_pointer** (const T ∗string1, const T ∗string2, const size_t string2Length)
- template<typename T >
  size_t **strncspn** (const T ∗stringToSearch, const size_t stringToSearchLength, const T ∗searchString, const size_t searchStringLength)
- template<typename T >
  size_t **find_last_not_of** (const T ∗string, const T ∗search, size_t offset=std::basic_string< T >::npos)
- template<typename T >
  size_t **find_last_of** (const T ∗string, const T ch, size_t offset=-1)
- template<typename T >
  size_t **find_first_not_of** (const T ∗stringToSearch, const size_t stringToSearchLength, const T ∗search↩String, const size_t searchStringLength)
- template<typename T >
  T **remove_all_whitespace** (const T &text)

  Removes all whitespace from a string.
- template<typename Tchar_type , typename T >
  void **remove_all** (T &text, Tchar_type char_to_replace)

  Removes all instances of a character from a string.
- template<typename Tchar_type , typename T >
  void **replace_all** (T &text, Tchar_type text_to_replace, Tchar_type replacement_text)

  helper functions for stemmers
- template<typename T , typename Tchar_type >
  void **replace_all** (T &text, const Tchar_type ∗text_to_replace, const Tchar_type ∗replacement_text)
- template<typename T >
  void **replace_all** (T &text, const T &text_to_replace, const T &replacement_text)
- template<typename string_typeT >
  size_t **remove_extra_spaces** (string_typeT &Text)
- template<typename string_typeT >
  size_t **remove_blank_lines** (string_typeT &Text)
- template<typename Tchar_type >
  double **strtod_ex** (const Tchar_type ∗nptr, Tchar_type ∗∗endptr)
- template<typename Tchar_type >
  bool **is_one_of** (const Tchar_type character, const Tchar_type ∗char_string)

6.2.1 Detailed Description

Date

2003-2015

Copyright

Oleander Software, Ltd.

Author

Oleander Software, Ltd.

This program is free software; you can redistribute it and/or modify it under the terms of the BSD License.

6.2.2 Function Documentation

6.2.2.1 template<typename T > int string_util::axtoi (const T ∗ *hexStr*, size_t *length* = -1) [inline]

Converts string in hex format to int. Default figures out how much of the string is a valid hex string, but passing a value to the second parameter overrides this and allows you to indicate how much of the string to try to convert.

Parameters

hexStr	The string to convert. How much of the string to analyze. The value -1 (the default) will tell the function to read until there are no more valid hexadecimal digits.

Returns

The value of the string as an integer.

6.2.2.2 template<typename T > size_t string_util::find_last_not_of (const T ∗ *string*, const T ∗ *search*, size_t *offset* = std::basic_string<T>::npos) [inline]

Searches for a single character not from a sequence in a string in reverse.

Parameters

string	The string to search in.
search	The sequence of characters to skip.
offset	Where to begin the search. If -1, then the reverse search will begin at the end of the string.

Returns

The position of where the last non-matching character is at, or -1 if it can't be found.

6.2.2.3 template<typename T > size_t string_util::find_last_of (const T ∗ *string*, const T *ch*, size_t *offset* = -1) [inline]

Searches for the last instance of a character in a string in reverse.

Parameters

string	The string to search.
ch	The character to search for.
offset	The offset in the string to begin the search from. The default (-1) will begin the search at the end of the string.

Returns

The offset of the found character, or -1 if not found.

6.2.2.4 template<typename T > const T* string_util::find_matching_close_tag (const T * *string*, const T *openSymbol*, const T *closeSymbol*, const bool *fail_on_overlapping_open_symbol* = `false`) `[inline]`

Searches for a matching tag, skipping an extra open/close pairs of symbols in between.

Parameters

openSymbol	The opening symbol.
closeSymbol	The closing symbol that we are looking for.
fail_on_overlapping_open_symbol	Whether it should immediately return failure if an open symbol is found before a matching close symbol.

Returns

A pointer to where the closing tag is, or NULL if one can't be found.

6.2.2.5 template<typename T > bool string_util::has_suffix (const T * *text*, const size_t *text_length*, const T * *suffix*, const size_t *suffix_length*) `[inline]`

Indicates whether a larger strings ends with the specified suffix. Lengths are provided by the caller for efficiency. This function is case sensitive.

6.2.2.6 template<typename T > bool string_util::is_hex_digit (const T *ch*) `[inline]`

Determines whether a character is a hexademical digit (0-9,A-F,a-f).

Parameters

ch	The letter to be analyzed.

6.2.2.7 template<typename T > bool string_util::is_space (const T *ch*) `[inline]`

Determines whether a character is a space, tab, or newline. Also includes double-width and no break spaces.

Parameters

ch	The letter to be analyzed.

6.2.2.8 template<typename string_typeT > size_t string_util::remove_blank_lines (string_typeT & *Text*)

Removes blank lines from a block of text

Parameters

Text	The text to have blank lines removed from.

Returns

Number of characters (not lines) removed from the block.

6.2.2.9 template<typename string_typeT > size_t string_util::remove_extra_spaces (string_typeT & *Text*)

Strips extraneous spaces/tabs/carriage returns from a block of text so that there isn't more than one space consecutively.

6.2.2.10 template<typename T > const T* string_util::strcspn_pointer (const T * *string1,* const T * *string2,* const size_t *string2Length*) `[inline]`

Searches for a single character from a sequence in a string and return a pointer if found.

6.2.2.11 template<typename T > int string_util::strnatordcmp (const T * *first_string,* const T * *second_string,* bool *case_insensitive* = `false`) `[inline]`

Natural order comparison functions. Compare, recognizing numeric strings.

6.2.2.12 template<typename T > const T* string_util::strnchr (const T * *string,* const T *ch,* size_t *numberOfCharacters*) `[inline]`

Searches for a single character in a string for n number of characters. Size argument should be less than or equal to the length of the string being searched.

Parameters

string	The string to search in.
ch	The character to search for.
numberOfCharacters	The number of characters to search through in the string.

Returns

A pointer in the string where the character was found, or NULL if not found.

6.2.2.13 template<typename T > size_t string_util::strncspn (const T * *stringToSearch,* const size_t *stringToSearchLength,* const T * *searchString,* const size_t *searchStringLength*) `[inline]`

Searches for a single character from a sequence in a string for n number of characters.

Parameters

stringToSearch	The string to search.
stringToSearchLength	The length of the string being searched.
searchString	The sequence of characters to search for.
searchStringLength	The length of the sequence string.

Returns

The index into the string that the character was found. Returns the length of the string if not found.

6.2.2.14 template< typename T > const T* string_util::strnistr (const T * *string*, const T * *strSearch*, const size_t *string_len*) [inline]

Searches for substring in a larger string (case-insensitively), limiting the search to a specified number of characters.

6.2.2.15 template< typename T > size_t string_util::strnlen (const T * *str*, const size_t *maxlen*) [inline]

Returns

The number of characters in the string pointed to by str, not including the terminating '\0' character, but at most maxlen. In doing this, strnlen looks only at the first maxlen characters at str and never beyond str+maxlen. This function should be used for input that may not be NULL terminated.

Parameters

str	The string to review.
maxlen	The maximum length of the string to scan.

Returns

The valid length of the string or maxlen, whichever is shorter.

6.2.2.16 template< typename T > const T* string_util::strrstr (const T * *string*, const T * *search*, size_t *offset*) [inline]

Search string in reverse for substring. "offset" is how far we are in the source string already and how far to go back.

6.2.2.17 template< typename Tchar_type > double string_util::strtod_ex (const Tchar_type * *nptr*, Tchar_type ** *endptr*) [inline]

Converts strings to double values, but also takes into account ranges (returning the average). For example, a string like "5-8" will return 6.5. Hyphens and colons are seen as range separators.

6.2.2.18 double string_util::strtol (const char * *str*, char ** *strend*, int *radix*) [inline]

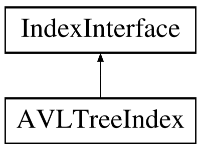
ANSI C decorators strtol

Chapter 7

Class Documentation

7.1 AVLTreeIndex Class Reference

Inheritance diagram for AVLTreeIndex:



Public Member Functions

- virtual vector$<$ **document** $>$ $*$ **findIndex** (string key)
- virtual void **addIndex** (string &word, string &doc)
- void **insert** (string x, string docname)
- void **insert** (string x, string docname, int count)
- void **remove** (string x)
- virtual void **display** ()
- virtual void **readIndex** ()
- virtual void **writeIndex** ()
- virtual vector$<$ **document** $>$ **topwords** ()
- virtual int **totalWordsIndexed** ()
- vector$<$ std::string $>$ **split** (const std::string &s, char delim)

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

- avltreeindex.h
- avltreeindex.cpp

7.2 backup_variable$<$ T $>$ Class Template Reference

class that remembers its original value from construction.

`#include <utilities.h>`

Public Member Functions

- **backup_variable** (const T &value)
- void **operator=** (const T &value)
- bool **operator==** (const T &value) const
- bool **operator<** (const T &value) const
- bool **operator<=** (const T &value) const
- bool **operator>** (const T &value) const
- bool **operator>=** (const T &value) const
- void **operator+** (const T &value)
- void **operator+=** (const T &value)
- void **operator-** (const T &value)
- void **operator-=** (const T &value)
- **operator const T** () const
- T ∗ **operator&** ()
- const T & **get_value** () const
- T & **get_value** ()
- bool **has_changed** () const

7.2.1 Detailed Description

template<typename T>
class backup_variable< T >

class that remembers its original value from construction.

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

- utilities.h

7.3 bookmark Struct Reference

Public Member Functions

- **bookmark** (string m, string q)

Public Attributes

- string **mark**
- string **query**

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

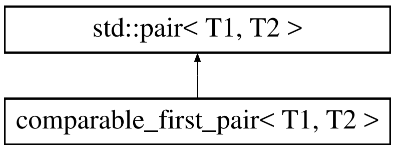
- searchengine.h

7.4 comparable_first_pair< T1, T2 > Class Template Reference

pair interface that compares on the first item

```
#include <utilities.h>
```

Inheritance diagram for comparable_first_pair< T1, T2 >:



Public Member Functions

- **comparable_first_pair** (const T1 &t1, const T2 &t2)
- bool **operator<** (const **comparable_first_pair**< T1, T2 > &that) const
- bool **operator==** (const **comparable_first_pair**< T1, T2 > &that) const

7.4.1 Detailed Description

template<typename T1, typename T2>
class comparable_first_pair< T1, T2 >

pair interface that compares on the first item

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

- utilities.h

7.5 document Struct Reference

Public Member Functions

- **document** (string name)
- **document** (string name, int count)
- string **getName** ()
- void **setName** (string name)
- void **setCount** (int num)
- int **getCount** ()

Public Attributes

- string **docname**
- int **count** =1
- double **tdif** =0

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

- indexinterface.h

7.6 DocumentParser Class Reference

Public Member Functions

- **DocumentParser** (**IndexHandler** ∗)
- string **getStemmed** (string &)
- bool **isStopWord** (string &)
- bool **extract** (string)
- int **getTotalPages** ()
- void **readInWordyMap** ()
- int **numOfDocs** ()
- void **clearWordTxt** ()
- bool **rawTextExtract** (string fileStream)

Public Attributes

- map< string, int > **wordy**
- vector< **TextExtractor** > **m**
- vector< string > **k**

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

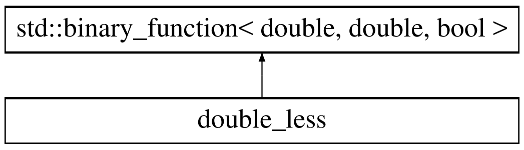
- DocumentParser.h
- DocumentParser.cpp

7.7 double_less Class Reference

"less" interface for double values.

```
#include <safe_math.h>
```

Inheritance diagram for double_less:



Public Member Functions

- bool **operator()** (const double &left, const double &right) const

7.7.1 Detailed Description

"less" interface for double values.

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

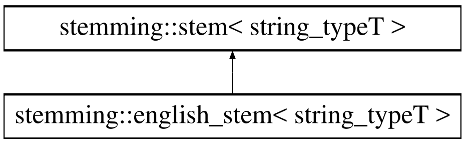
- safe_math.h

7.8 stemming::english_stem< string_typeT > Class Template Reference

English stemmer.

`#include <english_stem.h>`

Inheritance diagram for stemming::english_stem< string_typeT >:


stemming::stem< string_typeT >
stemming::english_stem< string_typeT >

Public Member Functions

- void **operator()** (string_typeT &text)

Additional Inherited Members

7.8.1 Detailed Description

template<typename string_typeT = std::wstring>
class stemming::english_stem< string_typeT >

English stemmer.

Overview:

I have made more than one attempt to improve the structure of the Porter algorithm by making it follow the pattern of ending removal of the Romance language stemmers. It is not hard to see why one should want to do this: step 1b of the Porter stemmer removes ed and ing, which are i-suffixes ($*$) attached to verbs. If these suffixes are removed, there should be no need to remove d-suffixes which are not verbal, although it will try to do so. This seems to be a deficiency in the Porter stemmer, not shared by the Romance stemmers. Again, the divisions between steps 2, 3 and 4 seem rather arbitrary, and are not found in the Romance stemmers. Nevertheless, these attempts at improvement have been abandoned. They seem to lead to a more complicated algorithm with no very obvious improvements. A reason for not taking note of the outcome of step 1b may be that English endings do not determine word categories quite as strongly as endings in the Romance languages. For example, condition and position in French have to be nouns, but in English they can be verbs as well as nouns, We are all conditioned by advertising They are positioning themselves differently today A possible reason for having separate steps 2, 3 and 4 is that d-suffix combinations in English are quite complex, a point which has been made elsewhere. But it is hardly surprising that after twenty years of use of the Porter stemmer, certain improvements do suggest themselves, and a new algorithm for English is therefore offered here. (It could be called the 'Porter2' stemmer to distinguish it from the Porter stemmer, from which it derives.) The changes are not so very extensive: (1) terminating y is changed to i rather less often, (2) suffix us does not lose its s, (3) a few additional suffixes are included for removal, including (4) suffix ly. In addition, a small list of exceptional forms is included. In December 2001 there were two further adjustments: (5) Steps 5a and 5b of the old Porter stemmer were combined into a single step. This means that undoubling final ll is not done with removal of final e. (6) In Step 3 ative is removed only when in region R2. To begin with, here is the basic algorithm without reference to the exceptional forms. An exact comparison with the Porter algorithm needs to be done quite carefully if done at all. Here we indicate by $*$ points of departure, and by + additional features. In the sample vocabulary, Porter and Porter2 stem slightly under 5% of words to different forms. Dr. Martin Porter Define a vowel as one of

- a e i o u y Define a double as one of
- bb dd ff gg mm nn pp rr tt Define a valid li-ending as one of
- c d e g h k m n r t Define a short syllable in a word as either (a) a vowel followed by a non-vowel other than w, x or Y and preceded by a non-vowel, or ∗ (b) a vowel at the beginning of the word followed by a non-vowel. So rap, trap, entrap end with a short syllable, and ow, on, at are classed as short syllables. But uproot, bestow, disturb do not end with a short syllable. A word is called short if it consists of a short syllable preceded by zero or more consonants. R1 is the region after the first non-vowel following a vowel, or the end of the word if there is no such non-vowel. R2 is the region after the first non-vowel following a vowel in R1, or the end of the word if there is no such non-vowel. If the word has two letters or less, leave it as it is. Otherwise, do each of the following operations, Set initial y, or y after a vowel, to Y, and then establish the regions R1 and R2.

Algorithm:

Step 1a: Search for the longest among the following suffixes, and perform the action indicated:

- sses
  - Replace by ss.
- ied+ ies∗
  - Replace by i if preceded by just one letter, otherwise by ie (so ties -> tie, cries -> cri).
- s
  - Delete if the preceding word part contains a vowel not immediately before the s (so gas and this retain the s, gaps and kiwis lose it).
- us+ ss
  - Do nothing. **Step 1b:** Search for the longest among the following suffixes, and perform the action indicated:
- eed eedly+
  - Replace by ee if in R1.
- ed edly+ ing ingly+
  - Delete if the preceding word part contains a vowel, and then
  - If the word ends at, bl or iz add e (so luxuriat -> luxuriate), or
  - If the word ends with a double remove the last letter (so hopp -> hop), or
  - If the word is short, add e (so hop -> hope). **Step 1c:** Replace suffix y or Y by i if preceded by a non-vowel which is not the first letter of the word (so cry -> cri, by -> by, say -> say) **Step 2:** Search for the longest among the following suffixes, and, if found and in R1, perform the action indicated:
- tional
  - Replace by tion.
- enci
  - Replace by ence.
- anci
  - Replace by ance
- abli
  - Replace by able.
- entli
  - Replace by ent.
- izer ization
  - Replace by ize.
- ational ation ator
  - Replace by ate.
- alism aliti alli

  - Replace by al.
- fulness
  - Replace by ful.
- ousli ousness
  - Replace by ous.
- iveness iviti
  - Replace by ive.
- biliti bli+
  - Replace by ble.
- ogi+
  - Replace by og if preceded by l.
- fulli+
  - Replace by ful.
- lessli+
  - Replace by less.
- li+
  - Delete if preceded by a valid li-ending. **Step 3:** Search for the longest among the following suffixes, and, if found and in R1, perform the action indicated:
- tional+
  - Replace by tion.
- ational+
  - Replace by ate.
- alize
  - Replace by al.
- icate iciti ical
  - Replace by ic.
- ful ness
  - Delete.
- ative$*$
  - Delete if in R2. **Step 4:** Search for the longest among the following suffixes, and, if found and in R2, perform the action indicated:
- al ance ence er ic able ible ant ement ment ent ism ate iti ous ive ize
  - Delete
- ion
  - Delete if preceded by s or t. **Step 5:** Search for the following suffixes, and, if found, perform the action indicated:
- e
  - Delete if in R2, or in R1 and not preceded by a short syllable.
- l
  - Delete if in R2 and preceded by l.

7.8.2 Member Function Documentation

7.8.2.1 template<typename string_typeT = std::wstring> void stemming::english_stem< string_typeT >::operator() (string_typeT & *text*) [inline]

Parameters

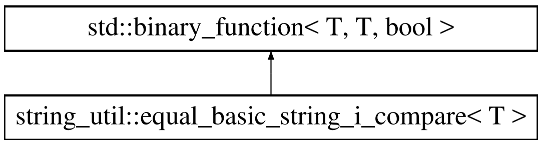
`in,out`	*text*	English string to stem.

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

- english_stem.h

7.9 string_util::equal_basic_string_i_compare< T > Class Template Reference

Inheritance diagram for string_util::equal_basic_string_i_compare< T >:


std::binary_function< T, T, bool >
string_util::equal_basic_string_i_compare< T >

Public Member Functions

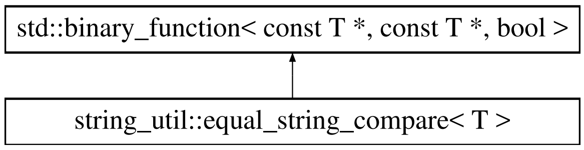
- bool **operator()** (const T &a_, const T &b_) const

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

- string_util.h

7.10 string_util::equal_string_compare< T > Class Template Reference

Inheritance diagram for string_util::equal_string_compare< T >:


std::binary_function< const T *, const T *, bool >
string_util::equal_string_compare< T >

Public Member Functions

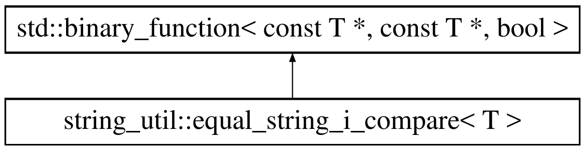
- bool **operator()** (const T ∗a_, const T ∗b_) const

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

- string_util.h

7.11 string_util::equal_string_i_compare< T > Class Template Reference

Inheritance diagram for string_util::equal_string_i_compare< T >:


std::binary_function< const T *, const T *, bool >
string_util::equal_string_i_compare< T >

Public Member Functions

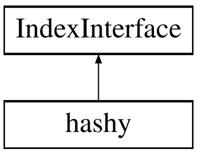
- bool **operator()** (const T ∗a_, const T ∗b_) const

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

- string_util.h

7.12 hashy Class Reference

Inheritance diagram for hashy:



Public Member Functions

- **hashy** ()

  This is the constructor for the custom hash table.
- int **Hashy** (string key)

  Hashy.
- void **addIndex** (string &word, string &name)
- int **NumItemsInIndex** (int index)
- void **display** ()
- void **displayIndex** (int index)
- vector< **document** > ∗ **findIndex** (string word)
- virtual void **readIndex** ()
- virtual void **writeIndex** ()
- virtual void **insertFromFile** (string &word, string &doc, int &count)
- vector< std::string > **split** (const std::string &s, char delim)
- virtual vector< **document** > **topwords** ()
- virtual int **totalWordsIndexed** ()

7.12.1 Member Function Documentation

7.12.1.1 int hashy::Hashy (string *key*)

Hashy.

Parameters

word	key

Returns

int

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

- hash.h
- hash.cpp

7.13 index_node Struct Reference

Public Attributes

- string **word_key**
- **index_node** $*$ **left**
- **index_node** $*$ **right**
- vector$<$ **document** $>$ **documents**
- int **height**

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

- avltreeindex.h

7.14 indexextractor Class Reference

Public Member Functions

- **indexextractor** (string)
- void **useStopWords** (string)
- bool **isStopWord** (string &)
- void **frequentTerm** (string, int)
- void **displayTermFrequency** ()
- void **allDocFrequency** (string word)
- void **corpusFrequency** (string)
- string **getStemmed** (string &)

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

- indexextractor.h
- indexextractor.cpp

7.15 IndexHandler Class Reference

Public Member Functions

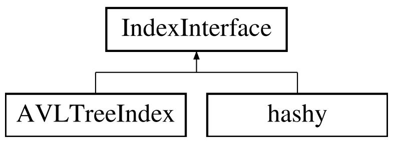
- **IndexHandler** (char type)
- void **writeIndex** ()
- void **addIndex** (string, string)
- vector< **document** > * **getDocs** (string)
- int **getDocumentFrequency** (string word, string docname)
- int **getCorpusFrequency** (string word)
- void **displayIndices** ()
- void **readIndex** ()
- vector< **document** > **topFifty** ()
- int **totalWordsIndexed** ()

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

- indexhandler.h
- indexhandler.cpp

7.16 IndexInterface Class Reference

Inheritance diagram for IndexInterface:



Public Member Functions

- virtual vector< **document** > * **findIndex** (string key)=0
- virtual void **addIndex** (string &word, string &doc)=0
- virtual void **display** ()=0
- virtual void **readIndex** ()=0
- virtual void **writeIndex** ()=0
- virtual vector< **document** > **topwords** ()=0
- virtual int **totalWordsIndexed** ()=0

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

- indexinterface.h

7.17 item Struct Reference

Public Attributes

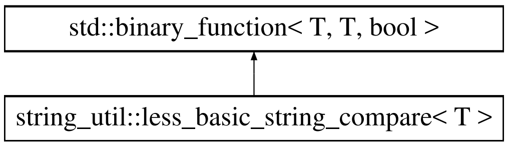
- string **word_key**
- int **count**
- string **docName**
- vector< **item** > **vec**

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

- hash.h

7.18 string_util::less_basic_string_compare< T > Class Template Reference

Inheritance diagram for string_util::less_basic_string_compare< T >:


std::binary_function< T, T, bool >
string_util::less_basic_string_compare< T >

Public Member Functions

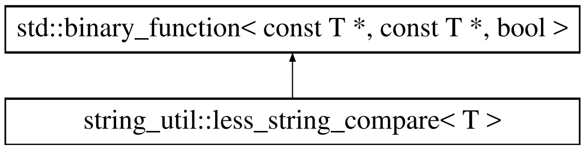
- bool **operator()** (const T &a_, const T &b_) const

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

- string_util.h

7.19 string_util::less_string_compare< T > Class Template Reference

Inheritance diagram for string_util::less_string_compare< T >:


std::binary_function< const T *, const T *, bool >
string_util::less_string_compare< T >

Public Member Functions

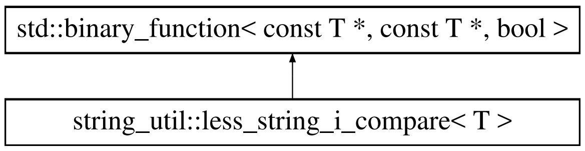
- bool **operator()** (const T ∗a_, const T ∗b_) const

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

- string_util.h

7.20 string_util::less_string_i_compare< T > Class Template Reference

Inheritance diagram for string_util::less_string_i_compare< T >:


std::binary_function< const T *, const T *, bool >
string_util::less_string_i_compare< T >

Public Member Functions

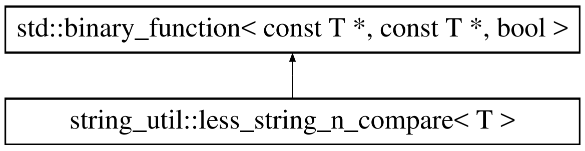
- bool **operator()** (const T ∗a_, const T ∗b_) const

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

- string_util.h

7.21 string_util::less_string_n_compare< T > Class Template Reference

Inheritance diagram for string_util::less_string_n_compare< T >:


std::binary_function< const T *, const T *, bool >
string_util::less_string_n_compare< T >

Public Member Functions

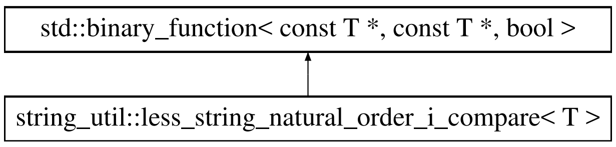
- **less_string_n_compare** (size_t comparison_size)
- bool **operator()** (const T ∗a_, const T ∗b_) const

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

- string_util.h

7.22 string_util::less_string_natural_order_i_compare< T > Class Template Reference

Inheritance diagram for string_util::less_string_natural_order_i_compare< T >:


std::binary_function< const T *, const T *, bool >
string_util::less_string_natural_order_i_compare< T >

Public Member Functions

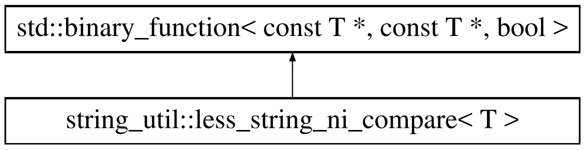
- bool **operator()** (const T ∗a_, const T ∗b_) const

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

- string_util.h

7.23 string_util::less_string_ni_compare< T > Class Template Reference

Inheritance diagram for string_util::less_string_ni_compare< T >:


std::binary_function< const T *, const T *, bool >
string_util::less_string_ni_compare< T >

Public Member Functions

- **less_string_ni_compare** (size_t comparison_size)
- bool **operator()** (const T ∗a_, const T ∗b_) const

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

- string_util.h

7.24 stemming::no_op_stem< string_typeT > Class Template Reference

Public Member Functions

- void **operator()** (const string_typeT &) const

  No-op stemming of declared string type.
- template<typename T >
  void **operator()** (const T &) const

  No-op stemming of flexible string type.

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

- stemming.h

7.25 QueryEngine Class Reference

Public Member Functions

- **QueryEngine** (**IndexHandler** ∗, **DocumentParser** ∗)
- vector< **document** > **querySearch** (string query)
- vector< **document** > **search** (string word)
- vector< **document** > **getIntersection** (vector< **document** > l, vector< **document** > r)
- vector< **document** > **getUnion** (vector< **document** > l, vector< **document** > r)
- vector< **document** > **getDifference** (vector< **document** > r, vector< **document** > l)
- void **findTDIFs** (vector< **document** > &currdocs)
- void **relevencySort** (vector< **document** > &currdocs)

Public Attributes

- int **fileSize** =0

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

- queryengine.h
- queryengine.cpp

7.26 rawOutputExtractor Class Reference

```
#include <rawoutputextractor.h>
```

Public Member Functions

- **rawOutputExtractor** (**IndexHandler** ∗, **indexextractor** ∗)
- int **getTotalPages** ()
- string **getDocName** ()
- int **getWordCount** ()
- void **Init** (const char ∗pszInput)

Public Attributes

- **IndexHandler** ∗ **ih**
- **indexextractor** ∗ **ie**
- string **mm**
- vector< string > **a**
- int **wordCount** =0
- string **docsName**

7.26.1 Detailed Description

This class uses the PoDoFo lib to parse a PDF file and to write all text it finds in this PDF document to stdout.

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

- rawoutputextractor.h
- rawoutputextractor.cpp

7.27 SearchEngine Class Reference

Public Member Functions

- **SearchEngine** (string docpath, char)
- vector< **document** > **display_search_results** (string word)
- void **relevencySort** (vector< **document** > &)
- void **writeIndex** ()
- void **readIndex** ()
- void **clearIndex** ()
- int **getTotalPages** ()
- int **numWordsIndexed** ()
- vector< **document** > **topfifty** ()
- bool **addDocumentsToIndex** (string docpath)
- void **chooseStructure** (char type)
- void **findTDIFs** (vector< **document** > &currdocs)
- int **readTotalPages** ()
- void **clearTotalPages** ()
- void **clearWordTxt** ()
- void **clear** ()
- bool **displayRawFile** (string filePath)
- void **displayTop50** ()
- void **writeFilePathToTXTFile** ()
- vector< string > **readFilePathFromTXTFile** ()
- void **clearFilePath** ()
- void **writeBookmark** (string book, string query)
- void **readBookmarks** ()
- void **addBookmark** (string book, string query)
- void **addToHistory** (string query)
- void **clearHistory** ()
- void **displayHistory** ()
- void **clearBookmarks** ()
- vector< **bookmark** > **displayBookmarks** ()

Public Attributes

- int **totalPages** =0

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

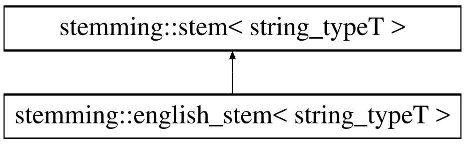
- searchengine.h
- searchengine.cpp

7.28 stemming::stem< string_typeT > Class Template Reference

The base class for language-specific stemmers.

```
#include <stemming.h>
```

Inheritance diagram for stemming::stem< string_typeT >:


stemming::stem< string_typeT >
stemming::english_stem< string_typeT >

Protected Member Functions

- void **find_r1** (const string_typeT &text, const wchar_t ∗vowel_list)
- void **find_r2** (const string_typeT &text, const wchar_t ∗vowel_list)
- void **find_spanish_rv** (const string_typeT &text, const wchar_t ∗vowel_list)
- void **find_french_rv** (const string_typeT &text, const wchar_t ∗vowel_list)
- void **find_russian_rv** (const string_typeT &text, const wchar_t ∗vowel_list)
- void **update_r_sections** (const string_typeT &text)
- bool **is_apostrophe** (const wchar_t &ch) const
- void **trim_western_punctuation** (string_typeT &text) const
- bool **is_suffix** (const string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U) const

  is_suffix for one character
- bool **is_suffix** (const string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar_↩ t suffix2L, const wchar_t suffix2U) const

  is_suffix for two characters
- bool **is_suffix** (const string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar_↩ t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U) const

  is_suffix for three characters
- bool **is_suffix** (const string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar_↩ t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U, const wchar_t suffix4L, const wchar_t suffix4U) const

  is_suffix for four characters
- bool **is_suffix** (const string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar_↩ t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U, const wchar_t suffix4L, const wchar_t suffix4U, const wchar_t suffix5L, const wchar_t suffix5U) const

  is_suffix for five characters
- bool **is_suffix** (const string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar_↩ t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U, const wchar_t suffix4L, const wchar_t suffix4U, const wchar_t suffix5L, const wchar_t suffix5U, const wchar_t suffix6L, const wchar↩ _t suffix6U) const

  is_suffix for six characters
- bool **is_suffix** (const string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar_↩ t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U, const wchar_t suffix4L, const wchar_t suffix4U, const wchar_t suffix5L, const wchar_t suffix5U, const wchar_t suffix6L, const wchar↩ _t suffix6U, const wchar_t suffix7L, const wchar_t suffix7U) const

  is_suffix for seven characters

- bool **is_suffix** (const string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar_←
t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U, const wchar_t suffix4L, const wchar_t suffix4U, const wchar_t suffix5L, const wchar_t suffix5U, const wchar_t suffix6L, const wchar←
_t suffix6U, const wchar_t suffix7L, const wchar_t suffix7U, const wchar_t suffix8L, const wchar_t suffix8U) const

    is_suffix for eight characters
- bool **is_suffix** (const string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar_←
t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U, const wchar_t suffix4L, const wchar_t suffix4U, const wchar_t suffix5L, const wchar_t suffix5U, const wchar_t suffix6L, const wchar←
_t suffix6U, const wchar_t suffix7L, const wchar_t suffix7U, const wchar_t suffix8L, const wchar_t suffix8U, const wchar_t suffix9L, const wchar_t suffix9U) const

    is_suffix for nine characters
- bool **is_partial_suffix** (const string_typeT &text, const size_t start_index, const wchar_t suffix1L, const wchar_t suffix1U, const wchar_t suffix2L, const wchar_t suffix2U)

    comparison for two characters
- bool **is_partial_suffix** (const string_typeT &text, const size_t start_index, const wchar_t suffix1L, const wchar_t suffix1U, const wchar_t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_←
t suffix3U)

    comparison for three characters
- bool **is_suffix_in_rv** (const string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U)

    RV suffix functions.
- bool **is_suffix_in_rv** (const string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar←
_t suffix2L, const wchar_t suffix2U)

    RV suffix comparison for two characters.
- bool **is_suffix_in_rv** (const string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar←
_t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U)

    RV suffix comparison for three characters.
- bool **is_suffix_in_rv** (const string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar←
_t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U, const wchar_t suffix4L, const wchar_t suffix4U)

    RV suffix comparison for four characters.
- bool **is_suffix_in_rv** (const string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar←
_t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U, const wchar_t suffix4L, const wchar_t suffix4U, const wchar_t suffix5L, const wchar_t suffix5U)

    RV suffix comparison for five characters.
- bool **is_suffix_in_rv** (const string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar←
_t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U, const wchar_t suffix4L, const wchar_t suffix4U, const wchar_t suffix5L, const wchar_t suffix5U, const wchar_t suffix6L, const wchar←
_t suffix6U)

    RV suffix comparison for six characters.
- bool **is_suffix_in_rv** (const string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar←
_t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U, const wchar_t suffix4L, const wchar_t suffix4U, const wchar_t suffix5L, const wchar_t suffix5U, const wchar_t suffix6L, const wchar←
_t suffix6U, const wchar_t suffix7L, const wchar_t suffix7U)

    RV suffix comparison for seven characters.
- bool **is_suffix_in_rv** (const string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar←
_t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U, const wchar_t suffix4L, const wchar_t suffix4U, const wchar_t suffix5L, const wchar_t suffix5U, const wchar_t suffix6L, const wchar←
_t suffix6U, const wchar_t suffix7L, const wchar_t suffix7U, const wchar_t suffix8L, const wchar_t suffix8U)

    RV suffix comparison for eight characters.
- bool **is_suffix_in_r1** (const string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U)

    R1 suffix functions.
- bool **is_suffix_in_r1** (const string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar←
_t suffix2L, const wchar_t suffix2U)

R1 suffix comparison for two characters.

- bool **is_suffix_in_r1** (const string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar←_t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U)

  R1 suffix comparison for three characters.

- bool **is_suffix_in_r1** (const string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar←_t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U, const wchar_t suffix4L, const wchar_t suffix4U)

  R1 suffix comparison for four characters.

- bool **is_suffix_in_r1** (const string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar←_t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U, const wchar_t suffix4L, const wchar_t suffix4U, const wchar_t suffix5L, const wchar_t suffix5U)

  R1 suffix comparison for five characters.

- bool **is_suffix_in_r1** (const string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar←_t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U, const wchar_t suffix4L, const wchar_t suffix4U, const wchar_t suffix5L, const wchar_t suffix5U, const wchar_t suffix6L, const wchar←_t suffix6U)

  R1 suffix comparison for six characters.

- bool **is_suffix_in_r2** (const string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U)

  R2 suffix functions.

- bool **is_suffix_in_r2** (const string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar←_t suffix2L, const wchar_t suffix2U)

  R2 suffix comparison for two characters.

- bool **is_suffix_in_r2** (const string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar←_t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U)

  R2 suffix comparison for three characters.

- bool **is_suffix_in_r2** (const string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar←_t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U, const wchar_t suffix4L, const wchar_t suffix4U)

  R2 suffix comparison for four characters.

- bool **is_suffix_in_r2** (const string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar←_t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U, const wchar_t suffix4L, const wchar_t suffix4U, const wchar_t suffix5L, const wchar_t suffix5U)

  R2 suffix comparison for five characters.

- bool **is_suffix_in_r2** (string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar_←t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U, const wchar_t suffix4L, const wchar_t suffix4U, const wchar_t suffix5L, const wchar_t suffix5U, const wchar_t suffix6L, const wchar←_t suffix6U)

  R2 suffix comparison for six characters.

- bool **is_suffix_in_r2** (const string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar←_t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U, const wchar_t suffix4L, const wchar_t suffix4U, const wchar_t suffix5L, const wchar_t suffix5U, const wchar_t suffix6L, const wchar←_t suffix6U, const wchar_t suffix7L, const wchar_t suffix7U)

  R2 suffix comparison for seven characters.

- bool **delete_if_is_in_r1** (string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const bool success_on_find=true)
- bool **delete_if_is_in_r1** (string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar_t suffix2L, const wchar_t suffix2U, const bool success_on_find=true)
- bool **delete_if_is_in_r1** (string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar_t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U, const bool success_on_←find=true)
- bool **delete_if_is_in_r1** (string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar_t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U, const wchar_t suffix4L, const wchar_t suffix4U, const bool success_on_find=true)

- bool **delete_if_is_in_r1** (string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar_t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U, const wchar_t suffix4L, const wchar_t suffix4U, const wchar_t suffix5L, const wchar_t suffix5U, const bool success_on_find=true)
- bool **delete_if_is_in_r1** (string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar↩ _t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U, const wchar_t suffix4L, const wchar_t suffix4U, const wchar_t suffix5L, const wchar_t suffix5U, const wchar_t suffix6L, const wchar↩ _t suffix6U, const bool success_on_find=true)
- bool **delete_if_is_in_r1** (string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar↩ _t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U, const wchar_t suffix4L, const wchar_t suffix4U, const wchar_t suffix5L, const wchar_t suffix5U, const wchar_t suffix6L, const wchar↩ _t suffix6U, const wchar_t suffix7L, const wchar_t suffix7U, const bool success_on_find=true)
- bool **delete_if_is_in_r2** (string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const bool success_on_find=true)
- bool **delete_if_is_in_r2** (string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar_t suffix2L, const wchar_t suffix2U, const bool success_on_find=true)
- bool **delete_if_is_in_r2** (string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar_t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U, const bool success_on_↩ find=true)
- bool **delete_if_is_in_r2** (string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar_t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U, const wchar_t suffix4L, const wchar_t suffix4U, const bool success_on_find=true)
- bool **delete_if_is_in_r2** (string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar_t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U, const wchar_t suffix4L, const wchar_t suffix4U, const wchar_t suffix5L, const wchar_t suffix5U, const bool success_on_find=true)

  R2 deletion for five character suffix.
- bool **delete_if_is_in_r2** (string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar↩ _t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U, const wchar_t suffix4L, const wchar_t suffix4U, const wchar_t suffix5L, const wchar_t suffix5U, const wchar_t suffix6L, const wchar↩ _t suffix6U, const bool success_on_find=true)

  R2 deletion for six character suffix.
- bool **delete_if_is_in_r2** (string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar↩ _t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U, const wchar_t suffix4L, const wchar_t suffix4U, const wchar_t suffix5L, const wchar_t suffix5U, const wchar_t suffix6L, const wchar↩ _t suffix6U, const wchar_t suffix7L, const wchar_t suffix7U, const bool success_on_find=true)

  R2 deletion for seven character suffix.
- bool **delete_if_is_in_r2** (string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar↩ _t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U, const wchar_t suffix4L, const wchar_t suffix4U, const wchar_t suffix5L, const wchar_t suffix5U, const wchar_t suffix6L, const wchar↩ _t suffix6U, const wchar_t suffix7L, const wchar_t suffix7U, const wchar_t suffix8L, const wchar_t suffix8U, const bool success_on_find=true)

  R2 deletion for eight character suffix.
- bool **delete_if_is_in_rv** (string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const bool success_on_find=true)
- bool **delete_if_is_in_rv** (string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar_t suffix2L, const wchar_t suffix2U, const bool success_on_find=true)
- bool **delete_if_is_in_rv** (string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar_t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U, const bool success_on_↩ find=true)
- bool **delete_if_is_in_rv** (string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar_t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U, const wchar_t suffix4L, const wchar_t suffix4U, const bool success_on_find=true)
- bool **delete_if_is_in_rv** (string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar_t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U, const wchar_t suffix4L, const wchar_t suffix4U, const wchar_t suffix5L, const wchar_t suffix5U, const bool success_on_find=true)

- bool **delete_if_is_in_rv** (string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar↩_t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U, const wchar_t suffix4L, const wchar_t suffix4U, const wchar_t suffix5L, const wchar_t suffix5U, const wchar_t suffix6L, const wchar↩_t suffix6U, const bool success_on_find=true)
- bool **delete_if_is_in_rv** (string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar↩_t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U, const wchar_t suffix4L, const wchar_t suffix4U, const wchar_t suffix5L, const wchar_t suffix5U, const wchar_t suffix6L, const wchar↩_t suffix6U, const wchar_t suffix7L, const wchar_t suffix7U, const bool success_on_find=true)
- bool **delete_if_is_in_rv** (string_typeT &text, const wchar_t suffix1L, const wchar_t suffix1U, const wchar↩_t suffix2L, const wchar_t suffix2U, const wchar_t suffix3L, const wchar_t suffix3U, const wchar_t suffix4L, const wchar_t suffix4U, const wchar_t suffix5L, const wchar_t suffix5U, const wchar_t suffix6L, const wchar↩_t suffix6U, const wchar_t suffix7L, const wchar_t suffix7U, const wchar_t suffix8L, const wchar_t suffix8U, const bool success_on_find=true)
- void **remove_german_umlauts** (string_typeT &text)
- void **italian_acutes_to_graves** (string_typeT &text)
- void **hash_dutch_yi** (string_typeT &text, const wchar_t ∗vowel_string)

  Hash initial y, y after a vowel, and i between vowels into hashed character.
- void **unhash_dutch_yi** (string_typeT &text)
- void **hash_german_yu** (string_typeT &text, const wchar_t ∗vowel_string)

  Hash 'u' and 'y' between vowels.
- void **unhash_german_yu** (string_typeT &text)
- void **hash_french_yui** (string_typeT &text, const wchar_t ∗vowel_string)
- void **unhash_french_yui** (string_typeT &text)
- void **hash_y** (string_typeT &text, const wchar_t ∗vowel_string)
- void **unhash_y** (string_typeT &text)
- void **hash_italian_ui** (string_typeT &text, const wchar_t ∗vowel_string)

  Hash u after q, and u, i between vowels.
- void **unhash_italian_ui** (string_typeT &text)
- void **remove_dutch_umlauts** (string_typeT &text)
- void **remove_dutch_acutes** (string_typeT &text)
- void **remove_spanish_acutes** (string_typeT &text)
- size_t **get_r1** () const
- void **set_r1** (const size_t val)
- size_t **get_r2** () const
- void **set_r2** (const size_t val)
- size_t **get_rv** () const
- void **set_rv** (const size_t val)
- void **reset_r_values** ()

7.28.1 Detailed Description

template<typename string_typeT = std::wstring>
class stemming::stem< string_typeT >

The base class for language-specific stemmers.

The template argument for the stemmers are the type of std::basic_string that you are trying to stem, by default std::wstring (Unicode strings). As long as the char type of your basic_string is wchar_t, then you can use any type of basic_string. This is to say, if your basic_string has a custom char_traits or allocator, then just specify it in your template argument to the stemmer.

Example:

```
typedef std::basic_string<wchar_t, myTraits, myAllocator> myString;
myString word(L"documentation");
stemming::english_stem<myString> StemEnglish;
StemEnglish(word);
```

7.28.2 Member Function Documentation

7.28.2.1 template<typename string_typeT = std::wstring> void stemming::stem< string_typeT >::hash_french_yui (string_typeT & *text*, const wchar_t ∗ *vowel_string*) `[inline]`, `[protected]`

Hash u or i preceded and followed by a vowel, and y preceded or followed by a vowel. u after q is also hashed. For example, jouer -> joUer ennuie -> ennuIe yeux -> Yeux quand -> qUand

7.28.2.2 template<typename string_typeT = std::wstring> bool stemming::stem< string_typeT >::is_apostrophe (const wchar_t & *ch*) const `[inline]`, `[protected]`

Determines if a character is an apostrophe (includes straight single quotes).

Parameters

ch	The letter to be analyzed.

7.28.2.3 template<typename string_typeT = std::wstring> bool stemming::stem< string_typeT >::is_suffix_in_r1 (const string_typeT & *text*, const wchar_t *suffix1L*, const wchar_t *suffix1U*) `[inline]`, `[protected]`

R1 suffix functions.

R1 suffix comparison for one character

7.28.2.4 template<typename string_typeT = std::wstring> bool stemming::stem< string_typeT >::is_suffix_in_r2 (const string_typeT & *text*, const wchar_t *suffix1L*, const wchar_t *suffix1U*) `[inline]`, `[protected]`

R2 suffix functions.

R2 suffix comparison for one character

7.28.2.5 template<typename string_typeT = std::wstring> bool stemming::stem< string_typeT >::is_suffix_in_rv (const string_typeT & *text*, const wchar_t *suffix1L*, const wchar_t *suffix1U*) `[inline]`, `[protected]`

RV suffix functions.

RV suffix comparison for one character

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

- stemming.h

7.29 string_util::string_tokenize< T > Class Template Reference

Tokenizes a string using a set of delimiters.

`#include <string_util.h>`

Public Member Functions

- **string_tokenize** (const T &val, const T &delim)
- bool **has_more_tokens** () const
- bool **has_more_delimiters** () const
- T **get_next_token** ()

7.29.1 Detailed Description

template<typename T>
class string_util::string_tokenize< T >

Tokenizes a string using a set of delimiters.

Date

2010

7.29.2 Constructor & Destructor Documentation

7.29.2.1 template<typename T > string_util::string_tokenize< T >::string_tokenize (const T & *val*, const T & *delim*) `[inline]`

Constructor which takes the string to parse and the delimiters to use.

Parameters

val	The string to parse.
delim	The set of delimiters to separate the string.

7.29.3 Member Function Documentation

7.29.3.1 template<typename T > T string_util::string_tokenize< T >::get_next_token ( ) `[inline]`

Returns

The next token from the original string as a string object Note that empty tokens can be returned if there is proceeding or trailing delimiters in the string, or if there are repeated delimiters next to each other.

7.29.3.2 template<typename T > bool string_util::string_tokenize< T >::has_more_delimiters ( ) const `[inline]`

Returns

Whether or not there are more delimiters in the string. This is useful for seeing if there are any delimiters at all when first loading the string.

7.29.3.3 template< typename T > bool string_util::string_tokenize< T >::has_more_tokens ( ) const [inline]

Returns

Whether or not there are more tokens in the string.

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

- string_util.h

7.30 string_util::string_trim< char_typeT > Class Template Reference

trims whitespace from around a string

`#include <string_util.h>`

Public Member Functions

- const char_typeT ∗ **operator()** (const char_typeT ∗value, size_t length=std::basic_string< char_typeT >↩ ::npos)
- size_t **get_trimmed_string_length** () const

7.30.1 Detailed Description

template<typename char_typeT>
class string_util::string_trim< char_typeT >

trims whitespace from around a string

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

- string_util.h

7.31 TextExtractor Class Reference

`#include <textextractor.h>`

Public Member Functions

- **TextExtractor** (**IndexHandler** ∗, **indexextractor** ∗)
- int **getTotalPages** ()
- string **getDocName** ()
- int **getWordCount** ()
- void **Init** (const char ∗pszInput, string docName)

Public Attributes

- **IndexHandler** ∗ **ih**
- **indexextractor** ∗ **ie**
- string **mm**
- vector< string > **a**
- int **wordCount** =0
- string **docsName**

7.31.1 Detailed Description

This class uses the PoDoFo lib to parse a PDF file and to write all text it finds in this PDF document to stdout.

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

- textextractor.h
- textextractor.cpp

7.32 userinterface Class Reference

Public Member Functions

- **userinterface** ()

  userinterface the object constructor
- void **use** ()

  the use method runs the user interface

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

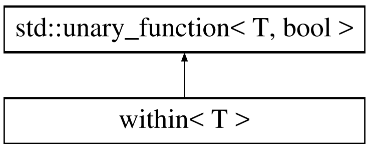
- userinterface.h
- userinterface.cpp

7.33 within< T > Class Template Reference

Determines if a value is within a given range.

```
#include <utilities.h>
```

Inheritance diagram for within< T >:



Public Member Functions

- **within** (T range_begin, T range_end)
- bool **operator()** (T value) const

7.33.1 Detailed Description

template$<$typename T$>$
class within$<$ T $>$

Determines if a value is within a given range.

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

- utilities.h

Index