NAME

ost::StringTokenizer −

Splits delimited string into tokens.

SYNOPSIS

```
#include <tokenizer.h>
```

Classes

class **iterator**
The input forward iterator for tokens.
class **NoSuchElementException**
Exception *thrown, if someone tried to read beyond the end of the tokens.*

Public Member Functions

StringTokenizer (const char *str, const char *delim, bool skipAllDelim=false, bool trim=false)
creates a new **StringTokenizer** *for a string and a given set of delimiters.*
StringTokenizer (const char *s)
create a new **StringTokenizer** *which splits the input string at whitespaces.*
iterator begin () const
returns the begin iterator
void **setDelimiters** (const char *d)
changes the set of delimiters used in subsequent iterations.
iterator begin (const char *d)
returns a begin iterator with an alternate set of delimiters.
const **iterator** & **end** () const
the iterator marking the end.

Static Public Attributes

static const char *const **SPACE**
a delimiter string containing all usual whitespace delimiters.

Friends

class **StringTokenizer::iterator**

Detailed Description

Splits delimited string into tokens.

The **StringTokenizer** takes a pointer to a string and a pointer to a string containing a number of possible delimiters. The **StringTokenizer** provides an input forward iterator which allows to iterate through all tokens. An iterator behaves like a logical pointer to the tokens, i.e. to shift to the next token, you’ve to increment the iterator, you get the token by dereferencing the iterator.

Memory consumption: This class operates on the original string and only allocates memory for the individual tokens actually requested, so this class allocates at maximum the space required for the longest token in the given string. Since for each iteration, memory is reclaimed for the last token, you MAY NOT store pointers to them; if you need them afterwards, copy them. You may not modify the original string while you operate on it with the **StringTokenizer**; the behaviour is undefined in that case.

The iterator has one special method ’nextDelimiter()’ which returns a character containing the next delimiter following this tokenization process or ’\0’, if there are no following delimiters. In case of skipAllDelim, it returns the FIRST delimiter.

With the method ’setDelimiters(const char*)’ you may change the set of delimiters. It affects all running iterators.

Example:

```
StringTokenizer st(’mary had a little lamb;its fleece was..’, ’ ;’);
StringTokenizer::iterator i;
for (i = st.begin() ; i != st.end() ; ++i) {
    cout << ’Token: ’’ << *i << ’’\t’;
    cout << ’ next Delim: ’’ << i.nextDelimiter() << ’’’ << endl;
}
```

Author:
Henner Zeller <H DOT Zeller AT acm DOT org>

License:.RS 4 LGPL

Constructor & Destructor Documentation

ost::StringTokenizer::StringTokenizer (const char * str, const char * delim, bool skipAllDelim = `false`**, bool trim =** `false`**)**

creates a new **StringTokenizer** for a string and a given set of delimiters. **Parameters:**

str **String** to be split up. This string will not be modified by this **StringTokenizer**, but you may as well not modfiy this string while tokenizing is in process, which may lead to undefined behaviour.
delim **String** containing the characters which should be regarded as delimiters.
skipAllDelim OPTIONAL. true, if subsequent delimiters should be skipped at once or false, if empty tokens should be returned for two delimiters with no other text inbetween. The first behaviour may be desirable for whitespace skipping, the second for input with delimited entry e.g. /etc/passwd like files or CSV input. NOTE, that 'true' here resembles the ANSI-C strtok(char *s,char *d) behaviour. DEFAULT = false
trim OPTIONAL. true, if the tokens returned should be trimmed, so that they don't have any whitespaces at the beginning or end. Whitespaces are any of the characters defined in **StringTokenizer::SPACE**. If delim itself is **StringTokenizer::SPACE**, this will result in a behaviour with skipAllDelim = true. DEFAULT = false

ost::StringTokenizer::StringTokenizer (const char * s)

create a new **StringTokenizer** which splits the input string at whitespaces. The tokens are stripped from whitespaces. This means, if you change the set of delimiters in either the 'begin(const char *delim)' method or in 'setDelimiters()', you then get whitespace trimmed tokens, delimited by the new set. Behaves like StringTokenizer(s, StringTokenizer::SPACE,false,true);

Member Function Documentation

iterator ost::StringTokenizer::begin (const char * d) `[inline]`

returns a begin iterator with an alternate set of delimiters.

iterator ost::StringTokenizer::begin () const `[inline]`

returns the begin iterator

const iterator& ost::StringTokenizer::end (void) const `[inline]`

the iterator marking the end.

void ost::StringTokenizer::setDelimiters (const char * d) `[inline]`

changes the set of delimiters used in subsequent iterations.

Friends And Related Function Documentation

friend class StringTokenizer::iterator `[friend]`

Member Data Documentation

const char* const ost::StringTokenizer::SPACE `[static]`

a delimiter string containing all usual whitespace delimiters. These are space, tab, newline, carriage return, formfeed and vertical tab. (see isspace() manpage).

Author

Generated automatically by Doxygen for GNU CommonC++ from the source code.