

CString

1.0

Generated by Doxygen 1.13.2

1 Class Index	1
1.1 Class List	1
2 File Index	3
2.1 File List	3
3 Class Documentation	5
3.1 CString Struct Reference	5
3.1.1 Detailed Description	5
3.1.2 Member Data Documentation	5
3.1.2.1 capacity	5
3.1.2.2 cs	6
3.1.2.3 data	6
3.1.2.4 length	6
4 File Documentation	7
4.1 include/cstr.h File Reference	7
4.1.1 Detailed Description	9
4.1.2 Macro Definition Documentation	9
4.1.2.1 CSTR_H	9
4.1.3 Function Documentation	10
4.1.3.1 cstr_append_chars()	10
4.1.3.2 cstr_append_cstr()	10
4.1.3.3 cstr_append_wchars()	10
4.1.3.4 cstr_at()	11
4.1.3.5 cstr_back()	11
4.1.3.6 cstr_capacity()	11
4.1.3.7 cstr_chars2wchars()	12
4.1.3.8 cstr_clear()	13
4.1.3.9 cstr_create()	13
4.1.3.10 cstr_create_from_buffer()	13
4.1.3.11 cstr_create_from_chars()	14
4.1.3.12 cstr_create_from_cstr()	14
4.1.3.13 cstr_create_from_wchars()	14
4.1.3.14 cstr_data()	15
4.1.3.15 cstr_destroy()	15
4.1.3.16 cstr_empty()	15
4.1.3.17 cstr_erase()	16
4.1.3.18 cstr_find_chars()	16
4.1.3.19 cstr_find_cstr()	16
4.1.3.20 cstr_find_wchars()	17
4.1.3.21 cstr_front()	17
4.1.3.22 cstr_get()	17

4.1.3.23	<code>cstr_insert()</code>	18
4.1.3.24	<code>cstr_length()</code>	18
4.1.3.25	<code>cstr_lock()</code>	18
4.1.3.26	<code>cstr_pop_back()</code>	19
4.1.3.27	<code>cstr_push_back_char()</code>	19
4.1.3.28	<code>cstr_push_back_wchar()</code>	19
4.1.3.29	<code>cstr_resize()</code>	20
4.1.3.30	<code>cstr_shrink_to_fit()</code>	20
4.1.3.31	<code>cstr_strdup()</code>	20
4.1.3.32	<code>cstr_substring()</code>	21
4.1.3.33	<code>cstr_swap()</code>	21
4.1.3.34	<code>cstr_to_lower()</code>	21
4.1.3.35	<code>cstr_to_upper()</code>	22
4.1.3.36	<code>cstr_tokenize()</code>	22
4.1.3.37	<code>cstr_tokenize_ex()</code>	22
4.1.3.38	<code>cstr_trim()</code>	23
4.1.3.39	<code>cstr_unlock()</code>	23
4.1.3.40	<code>cstr_wcsdup()</code>	23
4.1.4	Variable Documentation	24
4.1.4.1	<code>cstr_invalid</code>	24
4.2	<code>cstr.h</code>	24
Index		37

Chapter 1

Class Index

1.1 Class List

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

CString	Thread-safe dynamic string container	5
-------------------------	--	-------------------

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

include/ cstr.h	
Thread-safe dynamic string implementation for C	7

Chapter 3

Class Documentation

3.1 CString Struct Reference

Thread-safe dynamic string container.

```
#include <cstring.h>
```

Public Attributes

- char * [data](#)
Character buffer.
- size_t [length](#)
Current string length.
- size_t [capacity](#)
Allocated buffer size.
- CRITICAL_SECTION [cs](#)
Thread synchronization primitive.

3.1.1 Detailed Description

Thread-safe dynamic string container.

3.1.2 Member Data Documentation

3.1.2.1 capacity

```
size_t CString::capacity
```

Allocated buffer size.

3.1.2.2 cs

```
CRITICAL_SECTION CString::cs
```

Thread synchronization primitive.

3.1.2.3 data

```
char* CString::data
```

Character buffer.

3.1.2.4 length

```
size_t CString::length
```

Current string length.

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

- [include/cstr.h](#)

Chapter 4

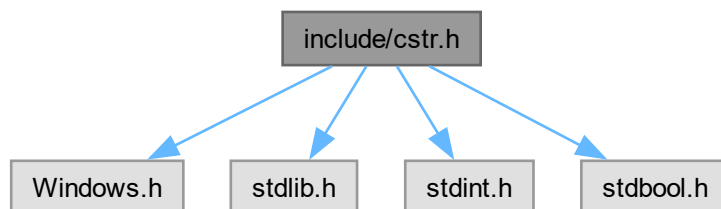
File Documentation

4.1 include/cstr.h File Reference

Thread-safe dynamic string implementation for C.

```
#include <Windows.h>
#include <stdlib.h>
#include <stdint.h>
#include <stdbool.h>
```

Include dependency graph for cstr.h:



Classes

- struct [CString](#)
Thread-safe dynamic string container.

Macros

- `#define` [CSTR_H](#)

Functions

- `char * cstr_strdup (_In_ const char *str)`
Duplicate null-terminated C string.
- `wchar_t * cstr_wcsdup (_In_ const wchar_t *str)`
Duplicate null-terminated wide string.
- `wchar_t * cstr_chars2wchars (_In_ const char *str, _In_ size_t cp)`
Convert multibyte string to wide character string.
- `bool cstr_create (_Inout_ CString *obj)`
Initialize a new empty [CString](#).
- `bool cstr_create_from_cstr (_Inout_ CString *obj, _In_ CString *obj2)`
Create [CString](#) copy from another [CString](#).
- `bool cstr_create_from_chars (_Inout_ CString *obj, _In_ const char *data)`
Create [CString](#) from null-terminated C string.
- `bool cstr_create_from_wchars (_Inout_ CString *obj, _In_ const wchar_t *data)`
Create [CString](#) from wide character string.
- `bool cstr_create_from_buffer (_Inout_ CString *obj, _In_ uint8_t *buffer, _In_ size_t size)`
Create [CString](#) from binary buffer.
- `bool cstr_destroy (_In_ CString *obj)`
Destroy [CString](#) and release resources.
- `void cstr_lock (_In_ CString *obj)`
Acquire exclusive access.
- `void cstr_unlock (_In_ CString *obj)`
Release exclusive access.
- `boolean cstr_at (_In_ CString *obj, _In_ size_t index, _Inout_ char *chr)`
Get character at specific index.
- `char cstr_get (_In_ CString *obj, _In_ size_t index)`
Direct character access (unsynchronized)
- `char cstr_front (_In_ CString *obj)`
Get first character.
- `char cstr_back (_In_ CString *obj)`
Get last character.
- `char * cstr_data (_In_ CString *obj)`
Get raw character buffer.
- `size_t cstr_length (_In_ CString *obj)`
Get current string length.
- `size_t cstr_capacity (_In_ CString *obj)`
Get allocated buffer capacity.
- `bool cstr_empty (_In_ CString *obj)`
Check if string is empty.
- `bool cstr_resize (_In_ CString *obj, _In_ size_t size)`
Resize internal buffer.
- `bool cstr_shrink_to_fit (_In_ CString *obj)`
Minimize buffer to fit current contents.
- `bool cstr_clear (_In_ CString *obj)`
Clear string contents.
- `bool cstr_push_back_char (_In_ CString *obj, _In_ char chr)`
Append single ASCII character.
- `bool cstr_push_back_wchar (_In_ CString *obj, _In_ wchar_t chr)`
Append wide character.
- `bool cstr_pop_back (_In_ CString *obj)`

- Remove last character.*
- bool `cstr_append_cstr` (`_In_ CString` *obj, `_In_ CString` *obj2)
- Append CString contents.*
- bool `cstr_append_chars` (`_In_ CString` *obj, `_In_ const char` *data)
- Append C string.*
- bool `cstr_append_wchars` (`_In_ CString` *obj, `_In_ const wchar_t` *data)
- Append wide string.*
- bool `cstr_substring` (`_In_ CString` *obj, `_Inout_ CString` *dest, `_In_ size_t` start, `_In_ size_t` length)
- Extract substring.*
- bool `cstr_erase` (`_In_ CString` *obj, `_In_ size_t` index, `_In_ size_t` size)
- Remove characters.*
- bool `cstr_insert` (`_In_ CString` *obj, `_In_ size_t` index, `_In_ char` chr)
- Insert character.*
- bool `cstr_swap` (`_In_ CString` *obj, `_In_ CString` *obj2)
- Swap contents between two CStrings.*
- `size_t` `cstr_find_cstr` (`_In_ CString` *obj, `_In_ CString` *obj2)
- Find substring (CString)*
- `size_t` `cstr_find_chars` (`_In_ CString` *obj, `_In_ const char` *data)
- Find substring (C string)*
- `size_t` `cstr_find_wchars` (`_In_ CString` *obj, `_In_ const wchar_t` *data)
- Find substring (wide string)*
- bool `cstr_to_upper` (`_In_ CString` *obj)
- Convert to uppercase.*
- bool `cstr_to_lower` (`_In_ CString` *obj)
- Convert to lowercase.*
- bool `cstr_trim` (`_In_ CString` *obj)
- Trim whitespace from both ends.*
- bool `cstr_tokenize` (`_In_ CString` *obj, `_Inout_ CString` *token, `_In_ const char` *delimiters, `_Inout_ size_t` *start_pos)
- Extract token using delimiters.*
- bool `cstr_tokenize_ex` (`_In_ CString` *obj, `_Inout_ CString` *token, `_In_ const char` *delimiters, `_In_ const char` *zone_pairs, `_In_ const char` *escape_chars, `_Inout_ size_t` *start_pos)
- Advanced tokenization with zones/escaping.*

Variables

- static const `size_t` `cstr_invalid` = (`size_t`)-1

4.1.1 Detailed Description

Thread-safe dynamic string implementation for C.

4.1.2 Macro Definition Documentation

4.1.2.1 CSTR_H

```
#define CSTR_H
```

4.1.3 Function Documentation

4.1.3.1 `cstr_append_chars()`

```
bool cstr_append_chars (  
    _In_ CString * obj,  
    _In_ const char * data)
```

Append C string.

Parameters

<i>obj</i>	Destination CString
<i>data</i>	Null-terminated source string

Returns

true on success

4.1.3.2 `cstr_append_cstr()`

```
bool cstr_append_cstr (  
    _In_ CString * obj,  
    _In_ CString * obj2)
```

Append [CString](#) contents.

Parameters

<i>obj</i>	Destination CString
<i>obj2</i>	Source CString

Returns

true on success

4.1.3.3 `cstr_append_wchars()`

```
bool cstr_append_wchars (  
    _In_ CString * obj,  
    _In_ const wchar_t * data)
```

Append wide string.

Parameters

<i>obj</i>	Destination CString
<i>data</i>	Null-terminated wide string

Returns

true on success

Note

Converts using system code page

4.1.3.4 cstr_at()

```
boolean cstr_at (
    _In_ CString * obj,
    _In_ size_t index,
    _Inout_ char * chr)
```

Get character at specific index.

Parameters

<i>obj</i>	CString object
<i>index</i>	Character position (0-based)
<i>chr</i>	Output character

Returns

true if index valid, false otherwise

Note

Thread-safe version with bounds checking

4.1.3.5 cstr_back()

```
char cstr_back (
    _In_ CString * obj)
```

Get last character.

Parameters

<i>obj</i>	CString object
------------	----------------

Returns

Last character or 0 if empty

4.1.3.6 cstr_capacity()

```
size_t cstr_capacity (
    _In_ CString * obj)
```

Get allocated buffer capacity.

Parameters

<i>obj</i>	CString object
------------	----------------

Returns

Capacity in bytes or CSTR_INVALID

4.1.3.7 `cstr_chars2wchars()`

```
wchar_t * cstr_chars2wchars (  
    _In_ const char * str,  
    _In_ size_t cp)
```

Convert multibyte string to wide character string.

Parameters

<i>str</i>	Null-terminated source multibyte string
<i>cp</i>	Character Page for conversion

Returns

New allocated wide string or NULL on failure

Warning

Caller must free result with free()

4.1.3.8 cstr_clear()

```
bool cstr_clear (
    _In_ CString * obj)
```

Clear string contents.

Parameters

<i>obj</i>	CString object
------------	----------------

Returns

true on success

Note

Securely erases buffer and resets length

4.1.3.9 cstr_create()

```
bool cstr_create (
    _Inout_ CString * obj)
```

Initialize a new empty CString.

Parameters

<i>obj</i>	Pointer to CString object to initialize
------------	---

Returns

true on success, false on allocation failure

Note

Creates empty string with capacity 1

4.1.3.10 cstr_create_from_buffer()

```
bool cstr_create_from_buffer (
    _Inout_ CString * obj,
    _In_ uint8_t * buffer,
    _In_ size_t size)
```

Create CString from binary buffer.

Parameters

<i>obj</i>	Destination CString
<i>buffer</i>	Source binary data
<i>size</i>	Number of bytes to copy

Returns

true on success, false on allocation failure

Note

Adds null-terminator after buffer contents

4.1.3.11 `cstr_create_from_chars()`

```
bool cstr_create_from_chars (  
    _Inout_ CString * obj,  
    _In_ const char * data)
```

Create [CString](#) from null-terminated C string.

Parameters

<i>obj</i>	Destination CString
<i>data</i>	Source C string

Returns

true on success, false on allocation failure

4.1.3.12 `cstr_create_from_cstr()`

```
bool cstr_create_from_cstr (  
    _Inout_ CString * obj,  
    _In_ CString * obj2)
```

Create [CString](#) copy from another [CString](#).

Parameters

<i>obj</i>	Destination CString
<i>obj2</i>	Source CString

Returns

true on success, false on allocation failure

4.1.3.13 `cstr_create_from_wchars()`

```
bool cstr_create_from_wchars (  
    _Inout_ CString * obj,  
    _In_ const wchar_t * data)
```

Create [CString](#) from wide character string.

Parameters

<i>obj</i>	Destination CString
<i>data</i>	Source wide string

Returns

true on success, false on conversion/allocation failure

Note

Uses WideCharToMultiByte with ANSI code page

4.1.3.14 cstr_data()

```
char * cstr_data (  
    _In_ CString * obj)
```

Get raw character buffer.

Parameters

<i>obj</i>	CString object
------------	--------------------------------

Returns

Pointer to internal buffer

Warning

Buffer valid until next modifying operation

4.1.3.15 cstr_destroy()

```
bool cstr_destroy (  
    _In_ CString * obj)
```

Destroy [CString](#) and release resources.

Parameters

<i>obj</i>	CString to destroy
------------	------------------------------------

Returns

true on success, false for invalid object

Note

Securely erases memory before freeing

4.1.3.16 cstr_empty()

```
bool cstr_empty (  
    _In_ CString * obj)
```

Check if string is empty.

Parameters

<i>obj</i>	CString object
------------	----------------

Returns

true if empty, false otherwise

4.1.3.17 cstr_erase()

```
bool cstr_erase (  
    _In_ CString * obj,  
    _In_ size_t index,  
    _In_ size_t size)
```

Remove characters.

Parameters

<i>obj</i>	CString object
<i>index</i>	Starting position
<i>size</i>	Number of characters to remove

Returns

true on success

4.1.3.18 cstr_find_chars()

```
size_t cstr_find_chars (  
    _In_ CString * obj,  
    _In_ const char * data)
```

Find substring (C string)

Parameters

<i>obj</i>	CString to search
<i>data</i>	Null-terminated substring

Returns

Starting index or CSTR_INVALID

4.1.3.19 cstr_find_cstr()

```
size_t cstr_find_cstr (  
    _In_ CString * obj,  
    _In_ CString * obj2)
```

Find substring (CString)

Parameters

<i>obj</i>	CString to search
<i>obj2</i>	Substring to find

Returns

Starting index or CSTR_INVALID

4.1.3.20 cstr_find_wchars()

```
size_t cstr_find_wchars (  
    _In_ CString * obj,  
    _In_ const wchar_t * data)
```

Find substring (wide string)

Parameters

<i>obj</i>	CString to search
<i>data</i>	Null-terminated wide substring

Returns

Starting index or CSTR_INVALID

Note

Converts using system code page

4.1.3.21 cstr_front()

```
char cstr_front (  
    _In_ CString * obj)
```

Get first character.

Parameters

<i>obj</i>	CString object
------------	----------------

Returns

First character or 0 if empty

4.1.3.22 cstr_get()

```
char cstr_get (  
    _In_ CString * obj,  
    _In_ size_t index)
```

Direct character access (unsynchronized)

Parameters

<i>obj</i>	CString object
<i>index</i>	Character position

Returns

Character or 0 for invalid index

Warning

Not thread-safe - use between lock/unlock calls

4.1.3.23 cstr_insert()

```
bool cstr_insert (  
    _In_ CString * obj,  
    _In_ size_t index,  
    _In_ char chr)
```

Insert character.

Parameters

<i>obj</i>	CString object
<i>index</i>	Insertion position
<i>chr</i>	Character to insert

Returns

true on success

4.1.3.24 cstr_length()

```
size_t cstr_length (  
    _In_ CString * obj)
```

Get current string length.

Parameters

<i>obj</i>	CString object
------------	----------------

Returns

Length in bytes or CSTR_INVALID

4.1.3.25 cstr_lock()

```
void cstr_lock (  
    _In_ CString * obj)
```

Acquire exclusive access.

Parameters

<i>obj</i>	CString object
------------	----------------

4.1.3.26 cstr_pop_back()

```
bool cstr_pop_back (  
    _In_ CString * obj)
```

Remove last character.

Parameters

<i>obj</i>	CString object
------------	----------------

Returns

true if character removed, false if empty

4.1.3.27 cstr_push_back_char()

```
bool cstr_push_back_char (  
    _In_ CString * obj,  
    _In_ char chr)
```

Append single ASCII character.

Parameters

<i>obj</i>	CString object
<i>chr</i>	Character to append

Returns

true on success

4.1.3.28 cstr_push_back_wchar()

```
bool cstr_push_back_wchar (  
    _In_ CString * obj,  
    _In_ wchar_t chr)
```

Append wide character.

Parameters

<i>obj</i>	CString object
<i>chr</i>	Wide character to append

Returns

true on success

Note

Converts to multibyte using system code page

4.1.3.29 cstr_resize()

```
bool cstr_resize (
    _In_ CString * obj,
    _In_ size_t size)
```

Resize internal buffer.

Parameters

<i>obj</i>	CString object
<i>size</i>	New buffer size

Returns

true on success, false on allocation failure

Note

Does not modify string contents

4.1.3.30 cstr_shrink_to_fit()

```
bool cstr_shrink_to_fit (
    _In_ CString * obj)
```

Minimize buffer to fit current contents.

Parameters

<i>obj</i>	CString object
------------	----------------

Returns

true on success, false on allocation failure

4.1.3.31 cstr_strdup()

```
char * cstr_strdup (
    _In_ const char * str)
```

Duplicate null-terminated C string.

Parameters

<i>str</i>	Source string to copy
------------	-----------------------

Returns

New allocated copy on success, NULL on failure

Note

Safe replacement for non-standard strdup()

Warning

Caller must free result with free()

4.1.3.32 cstr_substring()

```
bool cstr_substring (  
    _In_ CString * obj,  
    _Inout_ CString * dest,  
    _In_ size_t start,  
    _In_ size_t length)
```

Extract substring.

Parameters

<i>obj</i>	Source CString
<i>dest</i>	Destination CString
<i>start</i>	Starting index
<i>length</i>	Number of characters to extract

Returns

true on success

Note

Automatically clamps to valid range

4.1.3.33 cstr_swap()

```
bool cstr_swap (  
    _In_ CString * obj,  
    _In_ CString * obj2)
```

Swap contents between two CStrings.

Parameters

<i>obj</i>	First CString
<i>obj2</i>	Second CString

Returns

true on success

4.1.3.34 cstr_to_lower()

```
bool cstr_to_lower (  
    _In_ CString * obj)
```

Convert to lowercase.

Parameters

<i>obj</i>	CString object
------------	----------------

Returns

true on success

4.1.3.35 cstr_to_upper()

```
bool cstr_to_upper (
    _In_ CString * obj)
```

Convert to uppercase.

Parameters

<i>obj</i>	CString object
------------	----------------

Returns

true on success

4.1.3.36 cstr_tokenize()

```
bool cstr_tokenize (
    _In_ CString * obj,
    _Inout_ CString * token,
    _In_ const char * delimiters,
    _Inout_ size_t * start_pos)
```

Extract token using delimiters.

Parameters

<i>obj</i>	Source CString
<i>token</i>	Output token
<i>delimiters</i>	Separator characters
<i>start_pos</i>	Starting/ending position (updated)

Returns

true if token found

4.1.3.37 cstr_tokenize_ex()

```
bool cstr_tokenize_ex (
    _In_ CString * obj,
    _Inout_ CString * token,
    _In_ const char * delimiters,
    _In_ const char * zone_pairs,
    _In_ const char * escape_chars,
    _Inout_ size_t * start_pos)
```

Advanced tokenization with zones/escaping.

Parameters

<i>obj</i>	Source CString
<i>token</i>	Output token
<i>delimiters</i>	Separator characters
<i>zone_pairs</i>	Zone delimiter pairs (e.g., "\"")
<i>escape_chars</i>	Escape characters
<i>start_pos</i>	Starting/ending position (updated)

Returns

true if token found

```
size_t pos = 0;
CString str, token;
cstr_create_from_chars(&str, "Hello, \"my world\"!");
while (cstr_tokenize_ex(&str, &token, " ", "\"", "\\", &pos))
    printf("Token: %s\n", cstr_data(&token));
```

4.1.3.38 cstr_trim()

```
bool cstr_trim (
    _In_ CString * obj)
```

Trim whitespace from both ends.

Parameters

<i>obj</i>	CString object
------------	--------------------------------

Returns

true if modified, false otherwise

4.1.3.39 cstr_unlock()

```
void cstr_unlock (
    _In_ CString * obj)
```

Release exclusive access.

Parameters

<i>obj</i>	CString object
------------	--------------------------------

4.1.3.40 cstr_wcsdup()

```
wchar_t * cstr_wcsdup (
    _In_ const wchar_t * str)
```

Duplicate null-terminated wide string.

Parameters

<i>str</i>	Source wide string to copy
------------	----------------------------

Returns

New allocated copy on success, NULL on failure

Note

Wide char version of [cstr_strdup\(\)](#)

Warning

Caller must free result with `free()`

4.1.4 Variable Documentation**4.1.4.1 cstr_invalid**

```
const size_t cstr_invalid = (size_t)-1 [static]
```

4.2 cstr.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002
00007
00008 #ifndef CSTR_H
00009 #define CSTR_H
00010
00011 #include <Windows.h>
00012 #include <stdlib.h>
00013 #include <stdint.h>
00014 #include <stdbool.h>
00015
00016 #ifdef __cplusplus
00017 extern "C"
00018 {
00019 #endif
00020
00025     static const size_t cstr_invalid = (size_t)-1;
00026
00036     typedef struct
00037     {
00038         char* data;
00039         size_t length;
00040         size_t capacity;
00041         CRITICAL_SECTION cs;
00042     } CString;
00043
00051     char* cstr_strdup(_In_ const char* str)
00052     {
00053         size_t len = strlen(str) + 1;
00054         char* buf = (char*)malloc(len);
00055         if (buf)
00056             memcpy(buf, str, len);
00057         return buf;
00058     }
00059
00067     wchar_t* cstr_wcsdup(_In_ const wchar_t* str)
00068     {
00069         size_t len = (wcslen(str) + 1) * sizeof(wchar_t);
```

```

00070         wchar_t* buf = (wchar_t*)malloc(len);
00071         if (buf)
00072             memcpy(buf, str, len);
00073         return buf;
00074     }
00075
00083     wchar_t* cstr_chars2wchars(_In_ const char* str, _In_ size_t cp)
00084     {
00085         int wlen = MultiByteToWideChar(cp, MB_ERR_INVALID_CHARS, str, -1, NULL, 0);
00086         wchar_t* buf = (wchar_t*)malloc(wlen * sizeof(wchar_t));
00087         if (!buf)
00088             return NULL;
00089         if (!MultiByteToWideChar(cp, 0, str, -1, buf, wlen))
00090         {
00091             free(buf);
00092             return NULL;
00093         }
00094         return buf;
00095     }
00096
00103     bool cstr_create(_Inout_ CString* obj)
00104     {
00105         if (!obj)
00106             return false;
00107
00108         char* data = (char*)malloc(1);
00109         if (data == NULL)
00110             return false;
00111
00112         data[0] = '\\0';
00113
00114         obj->data = data;
00115         obj->length = 0;
00116         obj->capacity = 1;
00117
00118         InitializeCriticalSection(&obj->cs);
00119
00120         return true;
00121     }
00122
00129     bool cstr_create_from_cstr(_Inout_ CString* obj, _In_ CString* obj2)
00130     {
00131         if (!obj || !obj2)
00132             return false;
00133
00134         obj->data = cstr_strdup(obj2->data);
00135         obj->length = obj2->length;
00136         obj->capacity = obj2->capacity;
00137
00138         InitializeCriticalSection(&obj->cs);
00139
00140         return true;
00141     }
00142
00149     bool cstr_create_from_chars(_Inout_ CString* obj, _In_ const char* data)
00150     {
00151         if (!obj || !data)
00152             return false;
00153
00154         obj->data = cstr_strdup(data);
00155         obj->length = strlen(data);
00156         obj->capacity = obj->length + 1;
00157
00158         InitializeCriticalSection(&obj->cs);
00159
00160         return true;
00161     }
00162
00170     bool cstr_create_from_wchars(_Inout_ CString* obj, _In_ const wchar_t* data)
00171     {
00172         if (!obj || !data)
00173             return false;
00174
00175         int len = WideCharToMultiByte(CP_ACP, 0, data, -1, NULL, 0, NULL, NULL);
00176         if (len == 0)
00177             return false;
00178
00179         char* mb_data = (char*)malloc(len);
00180         if (!mb_data)
00181             return false;
00182
00183         if (!WideCharToMultiByte(CP_ACP, 0, data, -1, mb_data, len, NULL, NULL))
00184         {
00185             free(mb_data);
00186             return false;
00187         }
00188

```

```

00189         obj->data = mb_data;
00190         obj->length = strlen(mb_data);
00191         obj->capacity = len;
00192
00193         InitializeCriticalSection(&obj->cs);
00194
00195         return true;
00196     }
00197
00206 bool cstr_create_from_buffer(_Inout_ CString* obj, _In_ uint8_t* buffer, _In_ size_t size)
00207 {
00208     if (!obj || !buffer)
00209         return false;
00210
00211     char* data = (char*)malloc(size + 1);
00212     if (data == NULL)
00213         return false;
00214
00215     memcpy(data, (void*)buffer, size);
00216     data[size] = '\0';
00217
00218     obj->data = data;
00219     obj->length = size;
00220     obj->capacity = size + 1;
00221
00222     InitializeCriticalSection(&obj->cs);
00223
00224     return true;
00225 }
00226
00233 bool cstr_destroy(_In_ CString* obj)
00234 {
00235     if (!obj)
00236         return false;
00237
00238     if (obj->data)
00239     {
00240         SecureZeroMemory(obj->data, obj->capacity);
00241         free(obj->data);
00242         obj->data = NULL;
00243     }
00244
00245     DeleteCriticalSection(&obj->cs);
00246
00247     obj->length = 0;
00248     obj->capacity = 0;
00249
00250     return true;
00251 }
00252
00257 void cstr_lock(_In_ CString* obj)
00258 {
00259     if (obj)
00260         EnterCriticalSection(&obj->cs);
00261 }
00262
00267 void cstr_unlock(_In_ CString* obj)
00268 {
00269     if (obj)
00270         LeaveCriticalSection(&obj->cs);
00271 }
00272
00281 boolean cstr_at(_In_ CString* obj, _In_ size_t index, _Inout_ char* chr)
00282 {
00283     if (!obj)
00284         return false;
00285
00286     cstr_lock(obj);
00287
00288     if (index >= obj->length)
00289     {
00290         cstr_unlock(obj);
00291         return false;
00292     }
00293
00294     *chr = obj->data[index];
00295
00296     cstr_unlock(obj);
00297
00298     return true;
00299 }
00300
00308 char cstr_get(_In_ CString* obj, _In_ size_t index)
00309 {
00310     if (!obj)
00311         return 0;
00312

```

```
00313         cstr_lock(obj);
00314
00315         char out = obj->data[index];
00316
00317         cstr_unlock(obj);
00318
00319         return out;
00320     }
00321
00322     char cstr_front(_In_ CString* obj)
00323     {
00324         if (!obj)
00325             return 0;
00326
00327         cstr_lock(obj);
00328
00329         char out = cstr_get(obj, 0);
00330
00331         cstr_unlock(obj);
00332
00333         return out;
00334     }
00335
00336     char cstr_back(_In_ CString* obj)
00337     {
00338         if (!obj)
00339             return 0;
00340
00341         cstr_lock(obj);
00342
00343         char out = cstr_get(obj, obj->length - 1);
00344
00345         cstr_unlock(obj);
00346
00347         return out;
00348     }
00349
00350     char* cstr_data(_In_ CString* obj)
00351     {
00352         if (!obj)
00353             return 0;
00354
00355         cstr_lock(obj);
00356
00357         char* out = obj->data;
00358
00359         cstr_unlock(obj);
00360
00361         return out;
00362     }
00363
00364     size_t cstr_length(_In_ CString* obj)
00365     {
00366         if (!obj)
00367             return cstr_invalid;
00368
00369         cstr_lock(obj);
00370
00371         size_t out = obj->length;
00372
00373         cstr_unlock(obj);
00374
00375         return out;
00376     }
00377
00378     size_t cstr_capacity(_In_ CString* obj)
00379     {
00380         if (!obj)
00381             return cstr_invalid;
00382
00383         cstr_lock(obj);
00384
00385         size_t out = obj->capacity;
00386
00387         cstr_unlock(obj);
00388
00389         return out;
00390     }
00391
00392     bool cstr_empty(_In_ CString* obj)
00393     {
00394         if (!obj)
00395             return false;
00396
00397         cstr_lock(obj);
00398
00399         bool out = obj->data == NULL || obj->length == 0;
00400     }
```

```

00431
00432     cstr_unlock(obj);
00433
00434     return out;
00435 }
00436
00444 bool cstr_resize(_In_ CString* obj, _In_ size_t size)
00445 {
00446     if (!obj)
00447         return false;
00448
00449     cstr_lock(obj);
00450
00451     char* new_data = (char*)realloc(obj->data, size);
00452     if (new_data == NULL)
00453     {
00454         cstr_unlock(obj);
00455         return false;
00456     }
00457
00458     obj->data = new_data;
00459     obj->capacity = size;
00460
00461     cstr_unlock(obj);
00462
00463     return true;
00464 }
00465
00471 bool cstr_shrink_to_fit(_In_ CString* obj)
00472 {
00473     if (!obj)
00474         return false;
00475
00476     cstr_lock(obj);
00477
00478     if (!cstr_resize(obj, obj->length + 1))
00479     {
00480         cstr_unlock(obj);
00481         return false;
00482     }
00483
00484     cstr_unlock(obj);
00485
00486     return true;
00487 }
00488
00495 bool cstr_clear(_In_ CString* obj)
00496 {
00497     if (!obj)
00498         return false;
00499
00500     cstr_lock(obj);
00501
00502     SecureZeroMemory(obj->data, obj->capacity);
00503     obj->length = 0;
00504
00505     cstr_unlock(obj);
00506
00507     return true;
00508 }
00509
00516 bool cstr_push_back_char(_In_ CString* obj, _In_ char chr)
00517 {
00518     if (!obj)
00519         return false;
00520
00521     cstr_lock(obj);
00522
00523     if (obj->length + 1 >= obj->capacity)
00524     {
00525         if (!cstr_resize(obj, obj->length + 2))
00526         {
00527             cstr_unlock(obj);
00528             return false;
00529         }
00530     }
00531
00532     obj->data[obj->length] = chr;
00533     obj->data[obj->length + 1] = '\\0';
00534     obj->length++;
00535
00536     cstr_unlock(obj);
00537
00538     return true;
00539 }
00540
00548 bool cstr_push_back_wchar(_In_ CString* obj, _In_ wchar_t chr)

```



```

00549     {
00550         if (!obj)
00551             return false;
00552
00553         cstr_lock(obj);
00554
00555         wchar_t wstr[2] = { chr, L'\0' };
00556         int required_mb_len = WideCharToMultiByte(CP_ACP, 0, wstr, -1, NULL, 0, NULL, NULL);
00557         if (required_mb_len <= 0)
00558         {
00559             cstr_unlock(obj);
00560             return false;
00561         }
00562
00563         char* mb_str = (char*)malloc(required_mb_len);
00564         if (!mb_str)
00565         {
00566             cstr_unlock(obj);
00567             return false;
00568         }
00569
00570         if (WideCharToMultiByte(CP_ACP, 0, wstr, -1, mb_str, required_mb_len, NULL, NULL) == 0)
00571         {
00572             free(mb_str);
00573             cstr_unlock(obj);
00574             return false;
00575         }
00576
00577         size_t data_len = required_mb_len - 1;
00578
00579         size_t new_length = obj->length + data_len;
00580         size_t required_capacity = new_length + 1;
00581
00582         if (required_capacity > obj->capacity)
00583         {
00584             size_t new_capacity = required_capacity;
00585             char* new_data = (char*)realloc(obj->data, new_capacity);
00586             if (!new_data)
00587             {
00588                 free(mb_str);
00589                 cstr_unlock(obj);
00590                 return false;
00591             }
00592             obj->data = new_data;
00593             obj->capacity = new_capacity;
00594         }
00595
00596         memcpy(obj->data + obj->length, mb_str, data_len);
00597         obj->length = new_length;
00598         obj->data[new_length] = '\0';
00599
00600         free(mb_str);
00601         cstr_unlock(obj);
00602
00603         return true;
00604     }
00605
00611 bool cstr_pop_back(_In_ CString* obj)
00612 {
00613     if (!obj)
00614         return false;
00615
00616     cstr_lock(obj);
00617
00618     if (obj->length == 0)
00619     {
00620         cstr_unlock(obj);
00621         return false;
00622     }
00623
00624     obj->data[obj->length - 1] = 0;
00625     obj->length--;
00626
00627     cstr_unlock(obj);
00628
00629     return true;
00630 }
00631
00638 bool cstr_append_cstr(_In_ CString* obj, _In_ CString* obj2)
00639 {
00640     if (!obj || !obj2)
00641         return false;
00642
00643     cstr_lock(obj);
00644
00645     size_t new_length = obj->length + obj2->length;
00646     size_t required_capacity = new_length + 1;

```

```
00647
00648     if (required_capacity > obj->capacity)
00649     {
00650         if (!cstr_resize(obj, required_capacity))
00651         {
00652             cstr_unlock(obj);
00653             return false;
00654         }
00655     }
00656
00657     memcpy(obj->data + obj->length, obj2->data, obj2->length);
00658     obj->data[new_length] = '\0';
00659     obj->length = new_length;
00660
00661     cstr_unlock(obj);
00662
00663     return true;
00664 }
00665
00672 bool cstr_append_chars(_In_ CString* obj, _In_ const char* data)
00673 {
00674     if (!obj || !data)
00675         return false;
00676
00677     cstr_lock(obj);
00678
00679     size_t data_len = strlen(data);
00680     size_t new_length = obj->length + data_len;
00681     size_t required_capacity = new_length + 1;
00682
00683     if (required_capacity > obj->capacity)
00684     {
00685         if (!cstr_resize(obj, required_capacity))
00686         {
00687             cstr_unlock(obj);
00688             return false;
00689         }
00690     }
00691
00692     memcpy(obj->data + obj->length, data, data_len);
00693     obj->data[new_length] = '\0';
00694     obj->length = new_length;
00695
00696     cstr_unlock(obj);
00697
00698     return true;
00699 }
00700
00708 bool cstr_append_wchars(_In_ CString* obj, _In_ const wchar_t* data)
00709 {
00710     if (!obj || !data)
00711         return false;
00712
00713     cstr_lock(obj);
00714
00715     int len = WideCharToMultiByte(CP_ACP, 0, data, -1, NULL, 0, NULL, NULL);
00716     if (len == 0)
00717     {
00718         cstr_unlock(obj);
00719         return false;
00720     }
00721
00722     char* mb_data = (char*)malloc(len);
00723     if (!mb_data)
00724     {
00725         cstr_unlock(obj);
00726         return false;
00727     }
00728
00729     if (WideCharToMultiByte(CP_ACP, 0, data, -1, mb_data, len, NULL, NULL) == 0)
00730     {
00731         free(mb_data);
00732         cstr_unlock(obj);
00733         return false;
00734     }
00735
00736     size_t data_len = strlen(mb_data);
00737     size_t new_length = obj->length + data_len;
00738     size_t required_capacity = new_length + 1;
00739
00740     if (required_capacity > obj->capacity)
00741     {
00742         if (!cstr_resize(obj, required_capacity))
00743         {
00744             free(mb_data);
00745             cstr_unlock(obj);
00746             return false;
```

```

00747     }
00748 }
00749
00750 memcpy(obj->data + obj->length, mb_data, data_len);
00751 obj->data[new_length] = '\0';
00752 obj->length = new_length;
00753
00754 free(mb_data);
00755
00756 cstr_unlock(obj);
00757
00758 return true;
00759 }
00760
00770 bool cstr_substring(_In_ CString* obj, _Inout_ CString* dest, _In_ size_t start, _In_ size_t
length)
00771 {
00772     if (!obj || !dest)
00773         return false;
00774
00775     cstr_lock(obj);
00776
00777     if (start >= obj->length)
00778     {
00779         cstr_unlock(obj);
00780         return false;
00781     }
00782
00783     size_t max_length = obj->length - start;
00784     if (length > max_length)
00785         length = max_length;
00786
00787     char* buffer = (char*)malloc(length + 1);
00788     if (!buffer)
00789     {
00790         cstr_unlock(obj);
00791         return false;
00792     }
00793
00794     memcpy(buffer, obj->data + start, length);
00795     buffer[length] = '\0';
00796
00797     if (!cstr_create_from_chars(dest, buffer))
00798     {
00799         free(buffer);
00800         cstr_unlock(obj);
00801         return false;
00802     }
00803
00804     free(buffer);
00805
00806     cstr_unlock(obj);
00807
00808     return true;
00809 }
00810
00818 bool cstr_erase(_In_ CString* obj, _In_ size_t index, _In_ size_t size)
00819 {
00820     if (!obj)
00821         return false;
00822
00823     cstr_lock(obj);
00824
00825     if (index >= obj->length || size == 0)
00826     {
00827         cstr_unlock(obj);
00828         return false;
00829     }
00830
00831     if (size > obj->length - index)
00832         size = obj->length - index;
00833
00834     size_t new_length = obj->length - size;
00835     size_t move_size = (obj->length - (index + size)) + 1;
00836
00837     memmove(obj->data + index, obj->data + index + size, move_size);
00838     obj->length = new_length;
00839
00840     cstr_unlock(obj);
00841
00842     return true;
00843 }
00844
00852 bool cstr_insert(_In_ CString* obj, _In_ size_t index, _In_ char chr)
00853 {
00854     if (!obj)
00855         return false;

```

```

00856
00857     cstr_lock(obj);
00858
00859     if (index > obj->length)
00860     {
00861         cstr_unlock(obj);
00862         return false;
00863     }
00864
00865     size_t new_length = obj->length + 1;
00866     size_t required_capacity = new_length + 1;
00867
00868     if (required_capacity > obj->capacity)
00869     {
00870         if (!cstr_resize(obj, required_capacity))
00871         {
00872             cstr_unlock(obj);
00873             return false;
00874         }
00875     }
00876
00877
00878     memmove(obj->data + index + 1, obj->data + index, (obj->length - index) + 1);
00879     obj->data[index] = chr;
00880     obj->length = new_length;
00881
00882     cstr_unlock(obj);
00883
00884     return true;
00885 }
00886
00893 bool cstr_swap(_In_ CString* obj, _In_ CString* obj2)
00894 {
00895     if (!obj || !obj2)
00896         return false;
00897
00898     cstr_lock(obj);
00899     cstr_lock(obj2);
00900
00901     char* temp_data = obj->data;
00902     obj->data = obj2->data;
00903     obj2->data = temp_data;
00904
00905     size_t temp_length = obj->length;
00906     obj->length = obj2->length;
00907     obj2->length = temp_length;
00908
00909     size_t temp_capacity = obj->capacity;
00910     obj->capacity = obj2->capacity;
00911     obj2->capacity = temp_capacity;
00912
00913     cstr_unlock(obj2);
00914     cstr_unlock(obj);
00915
00916     return true;
00917 }
00918
00925 size_t cstr_find_cstr(_In_ CString* obj, _In_ CString* obj2)
00926 {
00927     if (!obj || !obj2)
00928         return cstr_invalid;
00929
00930     cstr_lock(obj);
00931     cstr_lock(obj2);
00932
00933     char* pos = strstr(obj->data, obj2->data);
00934     size_t out = (pos != NULL) ? (size_t)(pos - obj->data) : cstr_invalid;
00935
00936     cstr_unlock(obj2);
00937     cstr_unlock(obj);
00938
00939     return out;
00940 }
00941
00948 size_t cstr_find_chars(_In_ CString* obj, _In_ const char* data)
00949 {
00950     if (!obj || !data)
00951         return cstr_invalid;
00952
00953     cstr_lock(obj);
00954
00955     char* pos = strstr(obj->data, data);
00956     size_t out = (pos != NULL) ? (size_t)(pos - obj->data) : cstr_invalid;
00957
00958     cstr_unlock(obj);
00959
00960     return out;

```

```

00961     }
00962
00970 size_t cstr_find_wchars(_In_ CString* obj, _In_ const wchar_t* data)
00971 {
00972     if (!obj || !data)
00973         return cstr_invalid;
00974
00975     int len = WideCharToMultiByte(CP_ACP, 0, data, -1, NULL, 0, NULL, NULL);
00976     if (len == 0)
00977         return cstr_invalid;
00978
00979     char* mb_data = (char*)malloc(len);
00980     if (!mb_data)
00981         return cstr_invalid;
00982
00983     if (WideCharToMultiByte(CP_ACP, 0, data, -1, mb_data, len, NULL, NULL) == 0)
00984     {
00985         free(mb_data);
00986         return cstr_invalid;
00987     }
00988
00989     cstr_lock(obj);
00990
00991     char* pos = strstr(obj->data, mb_data);
00992     size_t result = (pos != NULL) ? (size_t)(pos - obj->data) : cstr_invalid;
00993
00994     cstr_unlock(obj);
00995
00996     free(mb_data);
00997
00998     return result;
00999 }
01000
01006 bool cstr_to_upper(_In_ CString* obj)
01007 {
01008     if (!obj)
01009         return false;
01010
01011     cstr_lock(obj);
01012
01013     for (size_t i = 0; i < obj->length; ++i)
01014         obj->data[i] = (char)toupper((unsigned char)obj->data[i]);
01015
01016     cstr_unlock(obj);
01017
01018     return true;
01019 }
01020
01026 bool cstr_to_lower(_In_ CString* obj)
01027 {
01028     if (!obj)
01029         return false;
01030
01031     cstr_lock(obj);
01032
01033     for (size_t i = 0; i < obj->length; ++i)
01034         obj->data[i] = (char)tolower((unsigned char)obj->data[i]);
01035
01036     cstr_unlock(obj);
01037
01038     return true;
01039 }
01040
01046 bool cstr_trim(_In_ CString* obj)
01047 {
01048     if (!obj)
01049         return false;
01050
01051     cstr_lock(obj);
01052
01053     if (obj->length == 0)
01054     {
01055         cstr_unlock(obj);
01056         return false;
01057     }
01058
01059     size_t start = 0;
01060     size_t end = obj->length - 1;
01061
01062     while (start <= end && isspace((unsigned char)obj->data[start]))
01063         start++;
01064
01065     while (end >= start && isspace((unsigned char)obj->data[end]))
01066         end--;
01067
01068     size_t new_length = (start <= end) ? (end - start + 1) : 0;
01069

```

```

01070         if (start > 0)
01071             memmove(obj->data, obj->data + start, new_length);
01072
01073         obj->data[new_length] = '\0';
01074         obj->length = new_length;
01075
01076         cstr_unlock(obj);
01077
01078         return true;
01079     }
01080
01081     bool cstr_tokenize(_In_ CString* obj, _Inout_ CString* token, _In_ const char* delimiters, _Inout_
size_t* start_pos)
01090     {
01091         if (!obj || !delimiters || !start_pos || *start_pos >= obj->length)
01092             return false;
01093
01094         cstr_lock(obj);
01095
01096         size_t len = obj->length;
01097         size_t pos = *start_pos;
01098
01099         while (pos < len && strchr(delimiters, obj->data[pos]) != NULL)
01100             pos++;
01101
01102         if (pos >= len)
01103         {
01104             *start_pos = pos;
01105             cstr_unlock(obj);
01106             return false;
01107         }
01108
01109         size_t token_start = pos;
01110
01111         while (pos < len && strchr(delimiters, obj->data[pos]) == NULL)
01112             pos++;
01113
01114         size_t token_end = pos;
01115
01116         size_t token_len = token_end - token_start;
01117         char* temp = (char*)malloc(token_len + 1);
01118         if (!temp)
01119         {
01120             cstr_unlock(obj);
01121             return false;
01122         }
01123
01124         memcpy(temp, obj->data + token_start, token_len);
01125         temp[token_len] = '\0';
01126
01127         if (!cstr_create_from_chars(token, temp))
01128         {
01129             free(temp);
01130             cstr_unlock(obj);
01131             return false;
01132         }
01133
01134         free(temp);
01135
01136         *start_pos = (token_end < len) ? token_end + 1 : len;
01137
01138         cstr_unlock(obj);
01139
01140         return true;
01141     }
01142
01143     bool cstr_tokenize_ex(_In_ CString* obj, _Inout_ CString* token, _In_ const char* delimiters, _In_
const char* zone_pairs, _In_ const char* escape_chars, _Inout_ size_t* start_pos)
01162     {
01163         if (!obj || !delimiters || !start_pos || *start_pos >= obj->length)
01164             return false;
01165
01166         cstr_lock(obj);
01167
01168         size_t len = obj->length;
01169         size_t pos = *start_pos;
01170
01171         while (pos < len && strchr(delimiters, obj->data[pos]) != NULL)
01172             pos++;
01173
01174         if (pos >= len)
01175         {
01176             *start_pos = pos;
01177             cstr_unlock(obj);
01178             return false;
01179         }
01180

```

```

01181     size_t token_start = pos;
01182     size_t token_end = cstr_invalid;
01183     bool in_zone = false;
01184     char zone_end = '\\0';
01185     bool escape = false;
01186
01187     for (; pos < len; pos++)
01188     {
01189         char c = obj->data[pos];
01190
01191         if (escape)
01192         {
01193             escape = false;
01194             continue;
01195         }
01196
01197         if (in_zone)
01198         {
01199             if (c == zone_end)
01200             {
01201                 in_zone = false;
01202                 zone_end = '\\0';
01203             }
01204         }
01205         else
01206         {
01207             if (strchr(delimiters, c) != NULL)
01208             {
01209                 token_end = pos;
01210                 break;
01211             }
01212
01213             if (zone_pairs)
01214             {
01215                 for (int z = 0; zone_pairs[z] != '\\0'; z += 2)
01216                 {
01217                     if (zone_pairs[z + 1] == '\\0')
01218                         break;
01219                     if (c == zone_pairs[z])
01220                     {
01221                         in_zone = true;
01222                         zone_end = zone_pairs[z + 1];
01223                         break;
01224                     }
01225                 }
01226             }
01227
01228             if (escape_chars && strchr(escape_chars, c) != NULL)
01229                 escape = true;
01230         }
01231     }
01232
01233     token_end = (pos == len) ? len : token_end;
01234
01235     size_t token_len = token_end - token_start;
01236     char* temp = (char*)malloc(token_len + 1);
01237     if (!temp)
01238     {
01239         cstr_unlock(obj);
01240         return false;
01241     }
01242
01243     memcpy(temp, obj->data + token_start, token_len);
01244     temp[token_len] = '\\0';
01245
01246     if (!cstr_create_from_chars(token, temp))
01247     {
01248         free(temp);
01249         cstr_unlock(obj);
01250         return false;
01251     }
01252
01253     free(temp);
01254     *start_pos = (token_end < len) ? token_end + 1 : len;
01255
01256     cstr_unlock(obj);
01257
01258     return true;
01259 }
01260
01261 #ifdef __cplusplus
01262 }
01263 #endif
01264
01265 #endif // CSTR_H

```


Index

- capacity
 - CString, [5](#)
- cs
 - CString, [5](#)
- cstr.h
 - cstr_append_chars, [10](#)
 - cstr_append_cstr, [10](#)
 - cstr_append_wchars, [10](#)
 - cstr_at, [10](#)
 - cstr_back, [11](#)
 - cstr_capacity, [11](#)
 - cstr_chars2wchars, [11](#)
 - cstr_clear, [13](#)
 - cstr_create, [13](#)
 - cstr_create_from_buffer, [13](#)
 - cstr_create_from_chars, [14](#)
 - cstr_create_from_cstr, [14](#)
 - cstr_create_from_wchars, [14](#)
 - cstr_data, [15](#)
 - cstr_destroy, [15](#)
 - cstr_empty, [15](#)
 - cstr_erase, [16](#)
 - cstr_find_chars, [16](#)
 - cstr_find_cstr, [16](#)
 - cstr_find_wchars, [17](#)
 - cstr_front, [17](#)
 - cstr_get, [17](#)
 - CSTR_H, [9](#)
 - cstr_insert, [18](#)
 - cstr_invalid, [24](#)
 - cstr_length, [18](#)
 - cstr_lock, [18](#)
 - cstr_pop_back, [19](#)
 - cstr_push_back_char, [19](#)
 - cstr_push_back_wchar, [19](#)
 - cstr_resize, [19](#)
 - cstr_shrink_to_fit, [20](#)
 - cstr_strdup, [20](#)
 - cstr_substring, [20](#)
 - cstr_swap, [21](#)
 - cstr_to_lower, [21](#)
 - cstr_to_upper, [22](#)
 - cstr_tokenize, [22](#)
 - cstr_tokenize_ex, [22](#)
 - cstr_trim, [23](#)
 - cstr_unlock, [23](#)
 - cstr_wcsdup, [23](#)
- cstr_append_chars
 - cstr.h, [10](#)
- cstr_append_cstr
 - cstr.h, [10](#)
- cstr_append_wchars
 - cstr.h, [10](#)
- cstr_at
 - cstr.h, [10](#)
- cstr_back
 - cstr.h, [11](#)
- cstr_capacity
 - cstr.h, [11](#)
- cstr_chars2wchars
 - cstr.h, [11](#)
- cstr_clear
 - cstr.h, [13](#)
- cstr_create
 - cstr.h, [13](#)
- cstr_create_from_buffer
 - cstr.h, [13](#)
- cstr_create_from_chars
 - cstr.h, [14](#)
- cstr_create_from_cstr
 - cstr.h, [14](#)
- cstr_create_from_wchars
 - cstr.h, [14](#)
- cstr_data
 - cstr.h, [15](#)
- cstr_destroy
 - cstr.h, [15](#)
- cstr_empty
 - cstr.h, [15](#)
- cstr_erase
 - cstr.h, [16](#)
- cstr_find_chars
 - cstr.h, [16](#)
- cstr_find_cstr
 - cstr.h, [16](#)
- cstr_find_wchars
 - cstr.h, [17](#)
- cstr_front
 - cstr.h, [17](#)
- cstr_get
 - cstr.h, [17](#)
- CSTR_H
 - cstr.h, [9](#)
- cstr_insert
 - cstr.h, [18](#)
- cstr_invalid
 - cstr.h, [24](#)
- cstr_length

- cstr.h, [18](#)
- cstr_lock
 - cstr.h, [18](#)
- cstr_pop_back
 - cstr.h, [19](#)
- cstr_push_back_char
 - cstr.h, [19](#)
- cstr_push_back_wchar
 - cstr.h, [19](#)
- cstr_resize
 - cstr.h, [19](#)
- cstr_shrink_to_fit
 - cstr.h, [20](#)
- cstr_strdup
 - cstr.h, [20](#)
- cstr_substring
 - cstr.h, [20](#)
- cstr_swap
 - cstr.h, [21](#)
- cstr_to_lower
 - cstr.h, [21](#)
- cstr_to_upper
 - cstr.h, [22](#)
- cstr_tokenize
 - cstr.h, [22](#)
- cstr_tokenize_ex
 - cstr.h, [22](#)
- cstr_trim
 - cstr.h, [23](#)
- cstr_unlock
 - cstr.h, [23](#)
- cstr_wcsdup
 - cstr.h, [23](#)
- CString, [5](#)
 - capacity, [5](#)
 - cs, [5](#)
 - data, [6](#)
 - length, [6](#)
- data
 - CString, [6](#)
- include/cstr.h, [7](#), [24](#)
- length
 - CString, [6](#)