CString

1.0

Generated by Doxygen 1.13.2

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 CString Struct Reference

Thread-safe dynamic string container.

`#include <cstr.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

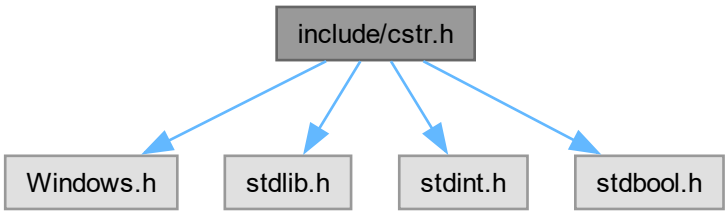
File Documentation

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.
- 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 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

obj	Destination CString
data	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

obj	Destination CString
obj2	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

obj	Destination CString
data	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

obj	CString object
index	Character position (0-based)
chr	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

obj	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

obj	CString object

Returns

Capacity in bytes or CSTR_INVALID

4.1.3.7 cstr_clear()

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

Clear string contents.

Parameters

obj	CString object

Returns

true on success

Note

Securely erases buffer and resets length

4.1.3.8 cstr_create()

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

Initialize a new empty CString.

Parameters

obj	Pointer to CString object to initialize

Returns

true on success, false on allocation failure

Note

Creates empty string with capacity 1

4.1.3.9 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

obj	Destination CString
buffer	Source binary data
size	Number of bytes to copy

Returns

true on success, false on allocation failure

Note

Adds null-terminator after buffer contents

4.1.3.10 cstr_create_from_chars()

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

Create CString from null-terminated C string.

Parameters

obj	Destination CString
data	Source C string

Returns

true on success, false on allocation failure

4.1.3.11 cstr_create_from_cstr()

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

Create CString copy from another CString.

Parameters

obj	Destination CString
obj2	Source CString

Returns

true on success, false on allocation failure

4.1.3.12 cstr_create_from_wchars()

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

Create CString from wide character string.

Parameters

obj	Destination CString
data	Source wide string

Returns

true on success, false on conversion/allocation failure

Note

Uses WideCharToMultiByte with ANSI code page

4.1.3.13 cstr_data()

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

Get raw character buffer.

Parameters

obj	CString object

Returns

Pointer to internal buffer

Warning

Buffer valid until next modifying operation

4.1.3.14 cstr_destroy()

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

Destroy CString and release resources.

Parameters

obj	CString to destroy

Returns

true on success, false for invalid object

Note

Securely erases memory before freeing

4.1.3.15 cstr_empty()

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

Check if string is empty.

Parameters

obj	CString object

Returns

true if empty, false otherwise

4.1.3.16 cstr_erase()

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

Remove characters.

Parameters

obj	CString object
index	Starting position
size	Number of characters to remove

Returns

true on success

4.1.3.17 cstr_find_chars()

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

Find substring (C string)

Parameters

obj	CString to search
data	Null-terminated substring

Returns

Starting index or CSTR_INVALID

4.1.3.18 cstr_find_cstr()

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

Find substring (CString)

Parameters

obj	CString to search
obj2	Substring to find

Returns

Starting index or CSTR_INVALID

4.1.3.19 cstr_find_wchars()

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

Find substring (wide string)

Parameters

obj	CString to search
data	Null-terminated wide substring

Returns

Starting index or CSTR_INVALID

Note

Converts using system code page

4.1.3.20 cstr_front()

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

Get first character.

Parameters

obj	CString object

Returns

First character or 0 if empty

4.1.3.21 cstr_get()

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

Direct character access (unsynchronized)

Parameters

obj	CString object
index	Character position

Returns

Character or 0 for invalid index

Warning

Not thread-safe - use between lock/unlock calls

4.1.3.22 cstr_insert()

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

Insert character.

Parameters

obj	CString object
index	Insertion position
chr	Character to insert

Returns

true on success

4.1.3.23 cstr_length()

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

Get current string length.

Parameters

obj	CString object

Returns

Length in bytes or CSTR_INVALID

4.1.3.24 cstr_lock()

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

Acquire exclusive access.

Parameters

obj	CString object

4.1.3.25 cstr_pop_back()

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

Remove last character.

Parameters

obj	CString object

Returns

true if character removed, false if empty

4.1.3.26 cstr_push_back_char()

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

Append single ASCII character.

Parameters

obj	CString object
chr	Character to append

Returns

true on success

4.1.3.27 cstr_push_back_wchar()

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

Append wide character.

Parameters

obj	CString object
chr	Wide character to append

Returns

true on success

Note

Converts to multibyte using system code page

4.1.3.28 cstr_resize()

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

Resize internal buffer.

Parameters

obj	CString object
size	New buffer size

Returns

true on success, false on allocation failure

Note

Does not modify string contents

4.1.3.29 cstr_shrink_to_fit()

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

Minimize buffer to fit current contents.

Parameters

obj	CString object

Returns

true on success, false on allocation failure

4.1.3.30 cstr_strdup()

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

Duplicate null-terminated C string.

Parameters

str	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()

4.1.3.31 cstr_substring()

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

Extract substring.

Parameters

obj	Source CString
dest	Destination CString
start	Starting index
length	Number of characters to extract

Returns

true on success

Note

Automatically clamps to valid range

4.1.3.32 cstr_swap()

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

Swap contents between two CStrings.

Parameters

obj	First CString
obj2	Second CString

Returns

true on success

4.1.3.33 cstr_to_lower()

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

Convert to lowercase.

Parameters

obj	CString object

Returns

true on success

4.1.3.34 cstr_to_upper()

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

Convert to uppercase.

Parameters

obj	CString object

Returns

true on success

4.1.3.35 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

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

Returns

true if token found

4.1.3.36 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

obj	Source CString
token	Output token
delimiters	Separator characters
zone_pairs	Zone delimiter pairs (e.g., "\"\"")
escape_chars	Escape characters
start_pos	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));
```

4.1.3.37 cstr_trim()

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

Trim whitespace from both ends.

Parameters

obj	CString object

Returns

true if modified, false otherwise

4.1.3.38 cstr_unlock()

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

Release exclusive access.

Parameters

obj	CString object

4.1.3.39 cstr_wcsdup()

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

Duplicate null-terminated wide string.

Parameters

str	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 invalid

```
const size_t 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 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
00082     bool cstr_create(_Inout_ CString* obj)
00083     {
00084         if (!obj)
00085             return false;
00086
00087         char* data = (char*)malloc(1);
00088         if (data == NULL)
00089             return false;
00090
00091         data[0] = '\0';
00092
00093         obj->data = data;
00094         obj->length = 0;
00095         obj->capacity = 1;
00096
00097         InitializeCriticalSection(&obj->cs);
00098
00099         return true;
00100     }
00101
00108     bool cstr_create_from_cstr(_Inout_ CString* obj, _In_ CString* obj2)
00109     {
00110         if (!obj || !obj2)
00111             return false;
```

```
00112
00113         obj->data = cstr_strdup(obj2->data);
00114         obj->length = obj2->length;
00115         obj->capacity = obj2->capacity;
00116
00117         InitializeCriticalSection(&obj->cs);
00118
00119         return true;
00120     }
00121
00128     bool cstr_create_from_chars(_Inout_ CString* obj, _In_ const char* data)
00129     {
00130         if (!obj || !data)
00131             return false;
00132
00133         obj->data = cstr_strdup(data);
00134         obj->length = strlen(data);
00135         obj->capacity = obj->length + 1;
00136
00137         InitializeCriticalSection(&obj->cs);
00138
00139         return true;
00140     }
00141
00149     bool cstr_create_from_wchars(_Inout_ CString* obj, _In_ const wchar_t* data)
00150     {
00151         if (!obj || !data)
00152             return false;
00153
00154         int len = WideCharToMultiByte(CP_ACP, 0, data, -1, NULL, 0, NULL, NULL);
00155         if (len == 0)
00156             return false;
00157
00158         char* mb_data = (char*)malloc(len);
00159         if (!mb_data)
00160             return false;
00161
00162         if (!WideCharToMultiByte(CP_ACP, 0, data, -1, mb_data, len, NULL, NULL))
00163         {
00164             free(mb_data);
00165             return false;
00166         }
00167
00168         obj->data = mb_data;
00169         obj->length = strlen(mb_data);
00170         obj->capacity = len;
00171
00172         InitializeCriticalSection(&obj->cs);
00173
00174         return true;
00175     }
00176
00185     bool cstr_create_from_buffer(_Inout_ CString* obj, _In_ uint8_t* buffer, _In_ size_t size)
00186     {
00187         if (!obj || !buffer)
00188             return false;
00189
00190         char* data = (char*)malloc(size + 1);
00191         if (data == NULL)
00192             return false;
00193
00194         memcpy(data, (void*)buffer, size);
00195         data[size] = '\0';
00196
00197         obj->data = data;
00198         obj->length = size;
00199         obj->capacity = size + 1;
00200
00201         InitializeCriticalSection(&obj->cs);
00202
00203         return true;
00204     }
00205
00212     bool cstr_destroy(_In_ CString* obj)
00213     {
00214         if (!obj)
00215             return false;
00216
00217         if (obj->data)
00218         {
00219             SecureZeroMemory(obj->data, obj->capacity);
00220             free(obj->data);
00221             obj->data = NULL;
00222         }
00223
00224         DeleteCriticalSection(&obj->cs);
00225
```

```
00226         obj->length = 0;
00227         obj->capacity = 0;
00228 
00229         return true;
00230     }
00231 
00236     void cstr_lock(_In_ CString* obj)
00237     {
00238         if (obj)
00239             EnterCriticalSection(&obj->cs);
00240     }
00241 
00246     void cstr_unlock(_In_ CString* obj)
00247     {
00248         if (obj)
00249             LeaveCriticalSection(&obj->cs);
00250     }
00251 
00260     boolean cstr_at(_In_ CString* obj, _In_ size_t index, _Inout_ char* chr)
00261     {
00262         if (!obj)
00263             return false;
00264 
00265         cstr_lock(obj);
00266 
00267         if (index >= obj->length)
00268         {
00269             cstr_unlock(obj);
00270             return false;
00271         }
00272 
00273         *chr = obj->data[index];
00274 
00275         cstr_unlock(obj);
00276 
00277         return true;
00278     }
00279 
00287     char cstr_get(_In_ CString* obj, _In_ size_t index)
00288     {
00289         if (!obj)
00290             return 0;
00291 
00292         cstr_lock(obj);
00293 
00294         char out = obj->data[index];
00295 
00296         cstr_unlock(obj);
00297 
00298         return out;
00299     }
00300 
00306     char cstr_front(_In_ CString* obj)
00307     {
00308         if (!obj)
00309             return 0;
00310 
00311         cstr_lock(obj);
00312 
00313         char out = cstr_get(obj, 0);
00314 
00315         cstr_unlock(obj);
00316 
00317         return out;
00318     }
00319 
00325     char cstr_back(_In_ CString* obj)
00326     {
00327         if (!obj)
00328             return 0;
00329 
00330         cstr_lock(obj);
00331 
00332         char out = cstr_get(obj, obj->length - 1);
00333 
00334         cstr_unlock(obj);
00335 
00336         return out;
00337     }
00338 
00345     char* cstr_data(_In_ CString* obj)
00346     {
00347         if (!obj)
00348             return 0;
00349 
00350         cstr_lock(obj);
00351 
```

```
00352         char* out = obj->data;
00353
00354         cstr_unlock(obj);
00355
00356         return out;
00357     }
00358
00364     size_t cstr_length(_In_ CString* obj)
00365     {
00366         if (!obj)
00367             return invalid;
00368
00369         cstr_lock(obj);
00370
00371         size_t out = obj->length;
00372
00373         cstr_unlock(obj);
00374
00375         return out;
00376     }
00377
00383     size_t cstr_capacity(_In_ CString* obj)
00384     {
00385         if (!obj)
00386             return invalid;
00387
00388         cstr_lock(obj);
00389
00390         size_t out = obj->capacity;
00391
00392         cstr_unlock(obj);
00393
00394         return out;
00395     }
00396
00402     bool cstr_empty(_In_ CString* obj)
00403     {
00404         if (!obj)
00405             return false;
00406
00407         cstr_lock(obj);
00408
00409         bool out = obj->data == NULL || obj->length == 0;
00410
00411         cstr_unlock(obj);
00412
00413         return out;
00414     }
00415
00423     bool cstr_resize(_In_ CString* obj, _In_ size_t size)
00424     {
00425         if (!obj)
00426             return false;
00427
00428         cstr_lock(obj);
00429
00430         char* new_data = (char*)realloc(obj->data, size);
00431         if (new_data == NULL)
00432         {
00433             cstr_unlock(obj);
00434             return false;
00435         }
00436
00437         obj->data = new_data;
00438         obj->capacity = size;
00439
00440         cstr_unlock(obj);
00441
00442         return true;
00443     }
00444
00450     bool cstr_shrink_to_fit(_In_ CString* obj)
00451     {
00452         if (!obj)
00453             return false;
00454
00455         cstr_lock(obj);
00456
00457         if (!cstr_resize(obj, obj->length + 1))
00458         {
00459             cstr_unlock(obj);
00460             return false;
00461         }
00462
00463         cstr_unlock(obj);
00464
00465         return true;
```

```
00466     }
00467
00474     bool cstr_clear(_In_ CString* obj)
00475     {
00476         if (!obj)
00477             return false;
00478
00479         cstr_lock(obj);
00480
00481         SecureZeroMemory(obj->data, obj->capacity);
00482         obj->length = 0;
00483
00484         cstr_unlock(obj);
00485
00486         return true;
00487     }
00488
00495     bool cstr_push_back_char(_In_ CString* obj, _In_ char chr)
00496     {
00497         if (!obj)
00498             return false;
00499
00500         cstr_lock(obj);
00501
00502         if (obj->length + 1 >= obj->capacity)
00503         {
00504             if (!cstr_resize(obj, obj->length + 2))
00505             {
00506                 cstr_unlock(obj);
00507                 return false;
00508             }
00509         }
00510
00511         obj->data[obj->length] = chr;
00512         obj->data[obj->length + 1] = '\0';
00513         obj->length++;
00514
00515         cstr_unlock(obj);
00516
00517         return true;
00518     }
00519
00527     bool cstr_push_back_wchar(_In_ CString* obj, _In_ wchar_t chr)
00528     {
00529         if (!obj)
00530             return false;
00531
00532         cstr_lock(obj);
00533
00534         wchar_t wstr[2] = { chr, L'\0' };
00535         int required_mb_len = WideCharToMultiByte(CP_ACP, 0, wstr, -1, NULL, 0, NULL, NULL);
00536         if (required_mb_len <= 0)
00537         {
00538             cstr_unlock(obj);
00539             return false;
00540         }
00541
00542         char* mb_str = (char*)malloc(required_mb_len);
00543         if (!mb_str)
00544         {
00545             cstr_unlock(obj);
00546             return false;
00547         }
00548
00549         if (WideCharToMultiByte(CP_ACP, 0, wstr, -1, mb_str, required_mb_len, NULL, NULL) == 0)
00550         {
00551             free(mb_str);
00552             cstr_unlock(obj);
00553             return false;
00554         }
00555
00556         size_t data_len = required_mb_len - 1;
00557
00558         size_t new_length = obj->length + data_len;
00559         size_t required_capacity = new_length + 1;
00560
00561         if (required_capacity > obj->capacity)
00562         {
00563             size_t new_capacity = required_capacity;
00564             char* new_data = (char*)realloc(obj->data, new_capacity);
00565             if (!new_data)
00566             {
00567                 free(mb_str);
00568                 cstr_unlock(obj);
00569                 return false;
00570             }
00571             obj->data = new_data;
```

```
00572             obj->capacity = new_capacity;
00573         }
00574
00575         memcpy(obj->data + obj->length, mb_str, data_len);
00576         obj->length = new_length;
00577         obj->data[new_length] = '\0';
00578
00579         free(mb_str);
00580         cstr_unlock(obj);
00581
00582         return true;
00583     }
00584
00590     bool cstr_pop_back(_In_ CString* obj)
00591     {
00592         if (!obj)
00593             return false;
00594
00595         cstr_lock(obj);
00596
00597         if (obj->length == 0)
00598         {
00599             cstr_unlock(obj);
00600             return false;
00601         }
00602
00603         obj->data[obj->length - 1] = 0;
00604         obj->length--;
00605
00606         cstr_unlock(obj);
00607
00608         return true;
00609     }
00610
00617     bool cstr_append_cstr(_In_ CString* obj, _In_ CString* obj2)
00618     {
00619         if (!obj || !obj2)
00620             return false;
00621
00622         cstr_lock(obj);
00623
00624         size_t new_length = obj->length + obj2->length;
00625         size_t required_capacity = new_length + 1;
00626
00627         if (required_capacity > obj->capacity)
00628         {
00629             if (!cstr_resize(obj, required_capacity))
00630             {
00631                 cstr_unlock(obj);
00632                 return false;
00633             }
00634         }
00635
00636         memcpy(obj->data + obj->length, obj2->data, obj2->length);
00637         obj->data[new_length] = '\0';
00638         obj->length = new_length;
00639
00640         cstr_unlock(obj);
00641
00642         return true;
00643     }
00644
00651     bool cstr_append_chars(_In_ CString* obj, _In_ const char* data)
00652     {
00653         if (!obj || !data)
00654             return false;
00655
00656         cstr_lock(obj);
00657
00658         size_t data_len = strlen(data);
00659         size_t new_length = obj->length + data_len;
00660         size_t required_capacity = new_length + 1;
00661
00662         if (required_capacity > obj->capacity)
00663         {
00664             if (!cstr_resize(obj, required_capacity))
00665             {
00666                 cstr_unlock(obj);
00667                 return false;
00668             }
00669         }
00670
00671         memcpy(obj->data + obj->length, data, data_len);
00672         obj->data[new_length] = '\0';
00673         obj->length = new_length;
00674
00675         cstr_unlock(obj);
```

```
00676
00677         return true;
00678     }
00679
00687     bool cstr_append_wchars(_In_ CString* obj, _In_ const wchar_t* data)
00688     {
00689         if (!obj || !data)
00690             return false;
00691
00692         cstr_lock(obj);
00693
00694         int len = WideCharToMultiByte(CP_ACP, 0, data, -1, NULL, 0, NULL, NULL);
00695         if (len == 0)
00696         {
00697             cstr_unlock(obj);
00698             return false;
00699         }
00700
00701         char* mb_data = (char*)malloc(len);
00702         if (!mb_data)
00703         {
00704             cstr_unlock(obj);
00705             return false;
00706         }
00707
00708         if (WideCharToMultiByte(CP_ACP, 0, data, -1, mb_data, len, NULL, NULL) == 0)
00709         {
00710             free(mb_data);
00711             cstr_unlock(obj);
00712             return false;
00713         }
00714
00715         size_t data_len = strlen(mb_data);
00716         size_t new_length = obj->length + data_len;
00717         size_t required_capacity = new_length + 1;
00718
00719         if (required_capacity > obj->capacity)
00720         {
00721             if (!cstr_resize(obj, required_capacity))
00722             {
00723                 free(mb_data);
00724                 cstr_unlock(obj);
00725                 return false;
00726             }
00727         }
00728
00729         memcpy(obj->data + obj->length, mb_data, data_len);
00730         obj->data[new_length] = '\0';
00731         obj->length = new_length;
00732
00733         free(mb_data);
00734
00735         cstr_unlock(obj);
00736
00737         return true;
00738     }
00739
00749     bool cstr_substring(_In_ CString* obj, _Inout_ CString* dest, _In_ size_t start, _In_ size_t
    length)
00750     {
00751         if (!obj || !dest)
00752             return false;
00753
00754         cstr_lock(obj);
00755
00756         if (start >= obj->length)
00757         {
00758             cstr_unlock(obj);
00759             return false;
00760         }
00761
00762         size_t max_length = obj->length - start;
00763         if (length > max_length)
00764             length = max_length;
00765
00766         char* buffer = (char*)malloc(length + 1);
00767         if (!buffer)
00768         {
00769             cstr_unlock(obj);
00770             return false;
00771         }
00772
00773         memcpy(buffer, obj->data + start, length);
00774         buffer[length] = '\0';
00775
00776         if (!cstr_create_from_chars(dest, buffer))
00777         {
```

```
00778             free(buffer);
00779             cstr_unlock(obj);
00780             return false;
00781         }
00782
00783         free(buffer);
00784
00785         cstr_unlock(obj);
00786
00787         return true;
00788     }
00789
00797     bool cstr_erase(_In_ CString* obj, _In_ size_t index, _In_ size_t size)
00798     {
00799         if (!obj)
00800             return false;
00801
00802         cstr_lock(obj);
00803
00804         if (index >= obj->length || size == 0)
00805         {
00806             cstr_unlock(obj);
00807             return false;
00808         }
00809
00810         if (size > obj->length - index)
00811             size = obj->length - index;
00812
00813         size_t new_length = obj->length - size;
00814         size_t move_size = (obj->length - (index + size)) + 1;
00815
00816         memmove(obj->data + index, obj->data + index + size, move_size);
00817         obj->length = new_length;
00818
00819         cstr_unlock(obj);
00820
00821         return true;
00822     }
00823
00831     bool cstr_insert(_In_ CString* obj, _In_ size_t index, _In_ char chr)
00832     {
00833         if (!obj)
00834             return false;
00835
00836         cstr_lock(obj);
00837
00838         if (index > obj->length)
00839         {
00840             cstr_unlock(obj);
00841             return false;
00842         }
00843
00844         size_t new_length = obj->length + 1;
00845         size_t required_capacity = new_length + 1;
00846
00847         if (required_capacity > obj->capacity)
00848         {
00849             if (!cstr_resize(obj, required_capacity))
00850             {
00851                 cstr_unlock(obj);
00852                 return false;
00853             }
00854         }
00855
00856
00857         memmove(obj->data + index + 1, obj->data + index, (obj->length - index) + 1);
00858         obj->data[index] = chr;
00859         obj->length = new_length;
00860
00861         cstr_unlock(obj);
00862
00863         return true;
00864     }
00865
00872     bool cstr_swap(_In_ CString* obj, _In_ CString* obj2)
00873     {
00874         if (!obj || !obj2)
00875             return false;
00876
00877         cstr_lock(obj);
00878         cstr_lock(obj2);
00879
00880         char* temp_data = obj->data;
00881         obj->data = obj2->data;
00882         obj2->data = temp_data;
00883
00884         size_t temp_length = obj->length;
```

```
00885         obj->length = obj2->length;
00886         obj2->length = temp_length;
00887
00888         size_t temp_capacity = obj->capacity;
00889         obj->capacity = obj2->capacity;
00890         obj2->capacity = temp_capacity;
00891
00892         cstr_unlock(obj2);
00893         cstr_unlock(obj);
00894
00895         return true;
00896     }
00897
00904     size_t cstr_find_cstr(_In_ CString* obj, _In_ CString* obj2)
00905     {
00906         if (!obj || !obj2)
00907             return invalid;
00908
00909         cstr_lock(obj);
00910         cstr_lock(obj2);
00911
00912         char* pos = strstr(obj->data, obj2->data);
00913         size_t out = (pos != NULL) ? (size_t)(pos - obj->data) : invalid;
00914
00915         cstr_unlock(obj2);
00916         cstr_unlock(obj);
00917
00918         return out;
00919     }
00920
00927     size_t cstr_find_chars(_In_ CString* obj, _In_ const char* data)
00928     {
00929         if (!obj || !data)
00930             return invalid;
00931
00932         cstr_lock(obj);
00933
00934         char* pos = strstr(obj->data, data);
00935         size_t out = (pos != NULL) ? (size_t)(pos - obj->data) : invalid;
00936
00937         cstr_unlock(obj);
00938
00939         return out;
00940     }
00941
00949     size_t cstr_find_wchars(_In_ CString* obj, _In_ const wchar_t* data)
00950     {
00951         if (!obj || !data)
00952             return invalid;
00953
00954         int len = WideCharToMultiByte(CP_ACP, 0, data, -1, NULL, 0, NULL, NULL);
00955         if (len == 0)
00956             return invalid;
00957
00958         char* mb_data = (char*)malloc(len);
00959         if (!mb_data)
00960             return invalid;
00961
00962         if (WideCharToMultiByte(CP_ACP, 0, data, -1, mb_data, len, NULL, NULL) == 0)
00963         {
00964             free(mb_data);
00965             return invalid;
00966         }
00967
00968         cstr_lock(obj);
00969
00970         char* pos = strstr(obj->data, mb_data);
00971         size_t result = (pos != NULL) ? (size_t)(pos - obj->data) : invalid;
00972
00973         cstr_unlock(obj);
00974
00975         free(mb_data);
00976
00977         return result;
00978     }
00979
00985     bool cstr_to_upper(_In_ CString* obj)
00986     {
00987         if (!obj)
00988             return false;
00989
00990         cstr_lock(obj);
00991
00992         for (size_t i = 0; i < obj->length; ++i)
00993             obj->data[i] = (char)toupper((unsigned char)obj->data[i]);
00994
00995         cstr_unlock(obj);
```

```
00996
00997         return true;
00998     }
00999
01005     bool cstr_to_lower(_In_ CString* obj)
01006     {
01007         if (!obj)
01008             return false;
01009
01010         cstr_lock(obj);
01011
01012         for (size_t i = 0; i < obj->length; ++i)
01013             obj->data[i] = (char)tolower((unsigned char)obj->data[i]);
01014
01015         cstr_unlock(obj);
01016
01017         return true;
01018     }
01019
01025     bool cstr_trim(_In_ CString* obj)
01026     {
01027         if (!obj)
01028             return false;
01029
01030         cstr_lock(obj);
01031
01032         if (obj->length == 0)
01033         {
01034             cstr_unlock(obj);
01035             return false;
01036         }
01037
01038         size_t start = 0;
01039         size_t end = obj->length - 1;
01040
01041         while (start <= end && isspace((unsigned char)obj->data[start]))
01042             start++;
01043
01044         while (end >= start && isspace((unsigned char)obj->data[end]))
01045             end--;
01046
01047         size_t new_length = (start <= end) ? (end - start + 1) : 0;
01048
01049         if (start > 0)
01050             memmove(obj->data, obj->data + start, new_length);
01051
01052         obj->data[new_length] = '\0';
01053         obj->length = new_length;
01054
01055         cstr_unlock(obj);
01056
01057         return true;
01058     }
01059
01068     bool cstr_tokenize(_In_ CString* obj, _Inout_ CString* token, _In_ const char* delimiters, _Inout_
      size_t* start_pos)
01069     {
01070         if (!obj || !delimiters || !start_pos || *start_pos >= obj->length)
01071             return false;
01072
01073         cstr_lock(obj);
01074
01075         size_t len = obj->length;
01076         size_t pos = *start_pos;
01077
01078         while (pos < len && strchr(delimiters, obj->data[pos]) != NULL)
01079             pos++;
01080
01081         if (pos >= len)
01082         {
01083             *start_pos = pos;
01084             cstr_unlock(obj);
01085             return false;
01086         }
01087
01088         size_t token_start = pos;
01089
01090         while (pos < len && strchr(delimiters, obj->data[pos]) == NULL)
01091             pos++;
01092
01093         size_t token_end = pos;
01094
01095         size_t token_len = token_end - token_start;
01096         char* temp = (char*)malloc(token_len + 1);
01097         if (!temp)
01098         {
01099             cstr_unlock(obj);
```

```
01100             return false;
01101         }
01102
01103         memcpy(temp, obj->data + token_start, token_len);
01104         temp[token_len] = '\0';
01105
01106         if (!cstr_create_from_chars(token, temp))
01107         {
01108             free(temp);
01109             cstr_unlock(obj);
01110             return false;
01111         }
01112
01113         free(temp);
01114
01115         *start_pos = (token_end < len) ? token_end + 1 : len;
01116
01117         cstr_unlock(obj);
01118
01119         return true;
01120     }
01121
01140     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)
01141     {
01142         if (!obj || !delimiters || !start_pos || *start_pos >= obj->length)
01143             return false;
01144
01145         cstr_lock(obj);
01146
01147         size_t len = obj->length;
01148         size_t pos = *start_pos;
01149
01150         while (pos < len && strchr(delimiters, obj->data[pos]) != NULL)
01151             pos++;
01152
01153         if (pos >= len)
01154         {
01155             *start_pos = pos;
01156             cstr_unlock(obj);
01157             return false;
01158         }
01159
01160         size_t token_start = pos;
01161         size_t token_end = invalid;
01162         bool in_zone = false;
01163         char zone_end = '\0';
01164         bool escape = false;
01165
01166         for (; pos < len; pos++)
01167         {
01168             char c = obj->data[pos];
01169
01170             if (escape)
01171             {
01172                 escape = false;
01173                 continue;
01174             }
01175
01176             if (in_zone)
01177             {
01178                 if (c == zone_end)
01179                 {
01180                     in_zone = false;
01181                     zone_end = '\0';
01182                 }
01183             }
01184             else
01185             {
01186                 if (strchr(delimiters, c) != NULL)
01187                 {
01188                     token_end = pos;
01189                     break;
01190                 }
01191
01192                 if (zone_pairs)
01193                 {
01194                     for (int z = 0; zone_pairs[z] != '\0'; z += 2)
01195                     {
01196                         if (zone_pairs[z + 1] == '\0')
01197                             break;
01198                         if (c == zone_pairs[z])
01199                         {
01200                             in_zone = true;
01201                             zone_end = zone_pairs[z + 1];
01202                             break;
01203                         }
```

```
01204                     }
01205                 }
01206
01207                 if (escape_chars && strchr(escape_chars, c) != NULL)
01208                     escape = true;
01209             }
01210         }
01211
01212         token_end = (pos == len) ? len : token_end;
01213
01214         size_t token_len = token_end - token_start;
01215         char* temp = (char*)malloc(token_len + 1);
01216         if (!temp)
01217         {
01218             cstr_unlock(obj);
01219             return false;
01220         }
01221
01222         memcpy(temp, obj->data + token_start, token_len);
01223         temp[token_len] = '\0';
01224
01225         if (!cstr_create_from_chars(token, temp))
01226         {
01227             free(temp);
01228             cstr_unlock(obj);
01229             return false;
01230         }
01231
01232         free(temp);
01233         *start_pos = (token_end < len) ? token_end + 1 : len;
01234
01235         cstr_unlock(obj);
01236
01237         return true;
01238     }
01239
01240 #ifdef __cplusplus
01241 }
01242 #endif
01243
01244 #endif // CSTR_H
```

Index