Documentation

Blib Docs

- [API](#)
  - [Buffers](#)
  - [Strings](#)
  - [Hashes](#)
  - [Dictionaries](#)
  - [Windows Functionality](#)

API

Buffers

This page contains the primitives and their associated functions for buffer primitives within blib. While the measured buffers support the valloc family of allocators, the Measured Buffer helper functions all use the halloc family, so halloc should be used where possible.

MeasuredBuffers

A general-purpose container for a defined number of MeasuredBuffers.

```
typedef struct _Container_MeasuredBuffer {
    DWORD count;
    MeasuredBuffer** members;
} Strings, MeasuredBuffers;
```

Initialisers

MeasuredBuffers* hallocMeasuredBuffers(DWORD count)

Allocates the measured buffers on the heap using the halloc family.

void hfreeMeasuredBuffers(MeasuredBuffers* buffers)

Frees the container and the subsequent pointers for each of the measured buffers allocated within the container IF the buffer is not NULL.

MeasuredBuffer

A general-purpose buffer with an assigned length up to `DWORD` bytes.

```
typedef struct _MeasuredBuffer {
    DWORD length;
    char* buffer;
```

```
} MeasuredBuffer, String;
```

Initialisers

MeasuredBuffer* vallocMeasuredBuffer(DWORD bytes)

Allocated a measured buffer `DWORD` bytes using `valloc`.

void vfreeMeasuredBuffer(MeasuredBuffer* buffer)

Frees a measured buffer AND its content buffer using the valloc family.

MeasuredBuffer* hallocMeasuredBuffer(DWORD bytes)

Allocated a measured buffer `DWORD` bytes using `halloc`.

void hfreeMeasuredBuffer(MeasuredBuffer* buffer);

Frees a measured buffer AND its content buffer using the halloc family.

MeasuredBuffer* bDeepCopyMeasuredBuffer(MeasuredBuffer* src)

Deep Copies a `Measured Buffer` 'src' and returns the pointer. This function uses the halloc family.

void bDeepCopyMeasuredBufferBuffer(MeasuredBuffer* dest, MeasuredBuffer* src);

Deep copies a `Measured Buffer` 'src' to the `Measured Buffer` 'dest'. This function allocates dest->buffer using the halloc family with the length of the original buffer. This frees the dest->buffer if it exists using halloc first.

Methods

BOOL bEncryptMeasuredBufferEx(void* algorithm, long* key, unsigned int keyLength, MeasuredBuffer* buffer)

Symmetrically encrypts or decrypts a `MeasuredBuffer` using the user supplied void `algorithm`.

DWORD bInterpetMeasuredBuffer(char* data, MeasuredBuffer* buffer)

Initialises and popualtes a measured buffer from data using `valloc`.

DWORD bWriteMeasuredBuffer(HANDLE hFile, MeasuredBuffer* buffer)

Writes a measured buffer `buffer` to a file `hFile`.

DWORD bReadMeasuredBuffer(HANDLE hFile, MeasuredBuffer* buffer)

Writes a measured buffer `buffer` to a file `hFile`.

CryptoBuffer

A 'cryptographic' primitive composed of two `MeasuredBuffer`s and a metadata element for an encryption type.

```
typedef enum _BLIB_ENCRYPTION_METHOD {
    BLIB_ENCRYPTION_NONE,
    BLIB_ENCRYPTION_UNDEFINED,
    BLIB_ENCRYPTION_STATIC_XOR,
} BLIB_ENCRYPTION_METHOD;


typedef struct _EncryptedBuffer{
    BLIB_ENCRYPTION_METHOD encryptionMethod;
    MeasuredBuffer keyBuffer;
    MeasuredBuffer dataBuffer;
} CryptoBuffer;
```

Methods

DWORD bSizeOfCryptoBuffer(CryptoBuffer* buffer)

Returns the `DWORD` size of the entire CryptoBuffer struct including all members and their respective buffers.

DWORD bInterpretCryptoBuffer(char* data, CryptoBuffer* buffer)

Interprets the memory located in `data` and populates the fields into `buffer` using bInterpretMeasuredBuffer.

DWORD bWriteCryptoBuffer(HANDLE hFile, CryptoBuffer* buffer)

Writes a `CryptoBuffer` to a file.

DWORD bReadCryptoBuffer(HANDLE hFile, CryptoBuffer* buffer)

Reads a `CryptoBuffer` to a file.

BOOL bEncryptBuffer(CryptoBuffer* buffer)

Symmetrically encrypts or decrypts a `CryptoBuffer` using the accompanying method defined in the encryptionMethod field of the `CryptoBuffer`.

BOOL bEncryptBufferEx(void* algorithm, CryptoBuffer* buffer)

Symmetrically encrypts or decrypts a `CryptoBuffer` using the user supplied void `algorithm`.

Strings

This page contains documentation about string types, including the hashing functions native to blib.

String

```
typedef struct _MeasuredBuffer {
    DWORD length;
    char* buffer;
} String, MeasuredBuffer;
```

Initialisers

String* vallocString(PBYTE text)

> This function creates a deepcopy of `text` and does not free the original buffer.

Uses `valloc` to allocate `String`.

String* hallocString(PBYTE text)

> This function creates a deepcopy of `text` and does not free the original buffer.

Uses `halloc` to allocate a `String`.

Methods

unsigned int rapidStringscmp(String* source, Strings* samples)

Returns the index of the sample `String` within the `Strings` that matches the `String` 'source'. Else returns -1 if none is found.

unsigned int rapidncmp (String source, unsigned int n, String* samples)

Compares a `String` `source` against an array of `String`. Returns the index of the matching `String`, or -1 if none are found.

unsigned int rapidstrcmp (String s1, String s2)

Compares two `String` s, `s1` and `s2`. Returns `0` if both strings are the same, or the first index if they differ. Returns `-1` if the strings are not of the same length.

Generic `char*` Functions

Methods

DWORD bSplitToStrings(PBYTE string, char c, MeasuredBuffers** buffers)

> This function is unsafe and may result in accessing illegal memory if there are no gaurd null-bytes on the source.

> This function returns up to a maximum of BLIB_MAXIMUM_SUBSTRINGS defined at the blib compile time. Additionally, this returns a maximum size of BLIB_MAXIMUM_SUBSTRING_SIZE for each of the given substrings. This function does NOT include the nullbyte and performs an in-place copy, so a substring of BLIB_MAXIMUM_SUBSTRING_SIZE length will NOT have a guard null byte.

Returns the number of substrings when splitting the input 'string' on the character 'c'. This populates the 'buffers' pointer with an array of MeasuredBuffers. An example usage is provided below where the Strings struct is used as the `MeasuredBuffer**` array.

```
Strings* strings;
bWideCharToByte(cStr, index);
DWORD dwStringCount = bSplitToStrings(cStr, '=', &strings);
```

```
if( dwStringCount == 1 ){
    hfreeMeasuredBuffers(strings);
    index += length;
    continue;
}
cprintf("%s : %s\n", strings->members[0]->buffer, strings->members[1]->buffer );
```

DWORD bSafeWideCharToByte(PBYTE dest, const PWCHAR source, PDWORD buffSize)

> This function is unsafe and may result in accessing illegal memory if there are no gaurd null-bytes on the source.

If the destination is `NULL` this functions returns the buffer size required in 'buffSize'. Otherwise, this function performs the widechar -> byte conversion and returns the length of the new buffer uszed in buffSize. This function adds and calculates the null byte '\0' in the destination.

DWORD bWideCharToByte(PBYTE dest, const PWCHAR source)

> This function is unsafe and may result in accessing illegal memory if there are no gaurd null-bytes on the source.

Converts a source wide char to the designated destination.

unsigned int bstrlen(PBYTE string)

> This function is unsafe and may result in accessing illegal memory if there are no gaurd null-bytes.

Returns the index of the first nullbyte.

Hashes

This page details the hashing functions for blib.

MiniHash

The Blib MiniHash is not cryptographically secure, but is a fast way to evaluate two arbitrary values where small collision chances are possible. Ideally, this can be used when searching for a string against a known string hash.

Methods

void blibMiniHashInit(DWORD seed)

Required function to initialise the blibMiniHash function with a seed `seed`.

DWORD blibMiniHash(PBYTE cstring)

> This function is dangerous. It assumes that there is a C-styled string with a guard nullbyte.

This function returns a `DWORD` using the following function:

```
DWORD blibMiniHash(PBYTE cstring){
    int index = 0;
    long returnValue = BLIB_MINIHASH_SEED;
    while(cstring[index] != '\0' && cstring[index] != '\n'){
        returnValue += (index * cstring[index]);
        index++;
    }
    return returnValue;
}
```

This function should be used only to validate that two things are 'similar' where exactness is not required. There is a low memory cost to using this.

DWORD blibMiniHashString(String* stirng)

This function performs the blibMiniHash using the seeded value.

Dictionaries

This page contains a definition for the MeasuredDictionary functionality within blib. This entity uses measuredbuffers for all members, including keys and values.

> Measured Dictionaries uses the halloc family to allocate on the heap. Keys are freed using hfree so values manually entered should be done using the halloc family.

> Measured Dictionaries will automatically resize at capacity. This requires calling hrealloc three times. At a future version this will be optimised to a single halloc call by stacking the keys, values, and tombstones arrays.

Measured Dictionary

```
typedef struct _MeasuredDict {
    DWORD count;
    DWORD capacity;
    WORD* tombstones;
    DWORD dwTombstones;
    MeasuredBuffer*    keys;
    MeasuredBuffer*    values;
} MeasuredDictionary, MD, Dictionary, Dict;
```

Initialisers

MeasuredDictionary* MDnew(DWORD initialCapcity)

Allocates the keys, values, and tombstones immediately on the heap up to initialCapacity.

void MDfree(MD* dict);

Frees the dictionary and its allocations.

Methods

BOOL MDadd(MD* dict,MeasuredBuffer* key, MeasuredBuffer* value)

Adds a `Measured Buffer` 'value' with a `Measured Buffer` 'key' to the dictionary dict. Returns `TRUE` on success.

MeasuredBuffer* MDget(MD* dict,MeasuredBuffer* key);

Returns a pointer to the `Measured Buffer` value that corrosponds to the `Measured Buffer` 'key'. Returns `NULL` on failure and sets the last error to `BLIB_ERROR_KEY_NOT_FOUND`.

BOOL MDset(MD* dict,MeasuredBuffer* key, MeasuredBuffer* value);

Sets the `Measured Buffer` 'value' to the `Measured Buffer` 'key' within dict. Returns `TRUE` on success, `FALSE` on fail. If this function fails, the error can be queried with bGetLastError. This should fail with `BLIB_ERROR_KEY_NOT_FOUND`.

BOOL MDremove(MD* dict,MeasuredBuffer* key)

> This function result in a double free at this point in time if the values are freed elsewhere in the code.

Unsets the key and frees the data fields of the key and value for the buffers pointed to by this element. Returns `TRUE` on success and `FALSE` on failure.

unsigned int MDcontainsKey(MD* dict, MeasuredBuffer* key);

Returns the index of the `Measured Buffer` key within the dict. Returns `-1` upon failure and sets the error code `BLIB_ERROR_KEY_NOT_FOUND`.

Windows Functionality

This page details some windows functionality for Blib processes.

```
#include <blibwin.h>
```

General

```
typedef struct _BLIB_ENV {
    unsigned int size;
    WCHAR* env;
} ENV, BLIB_ENV;
```

BLIB_ENV bGetEnv();

Returns a `BLIB_ENV` struct from the TEB->PEB.

UNICODE_STRING* blibGetCmdLine()

Returns a pointer tto the `UNICODE_STRING` within the PEB.