Ghoti.io CUtil

0.1

Generated by Doxygen 1.9.1

1 Ghoti.io CUtil Library 1

1.0.1 Overview . . . 1
1.0.2 Installation . . . 1
1.0.2.1 Build From Source . . . 1
1.0.3 Compiling With The Library . . . 1

2 Class Index 3

2.1 Class List . . . 3

3 File Index 5

3.1 File List . . . 5

4 Class Documentation 7

4.1 GCU_Hash_Cell Struct Reference . . . 7
4.1.1 Detailed Description . . . 8
4.2 GCU_Hash_Iterator Struct Reference . . . 8
4.2.1 Detailed Description . . . 9
4.3 GCU_Hash_Table Struct Reference . . . 9
4.3.1 Detailed Description . . . 10
4.4 GCU_Hash_Value Struct Reference . . . 10
4.4.1 Detailed Description . . . 11
4.5 GCU_Type_Union Union Reference . . . 11
4.5.1 Detailed Description . . . 11

5 File Documentation 13

5.1 include/cutil/debug.h File Reference . . . 13
5.1.1 Detailed Description . . . 14
5.1.2 Function Documentation . . . 14
5.1.2.1 gcu_calloc() . . . 14
5.1.2.2 gcu_free() . . . 15
5.1.2.3 gcu_malloc() . . . 15
5.1.2.4 gcu_realloc() . . . 15
5.2 include/cutil/hash.h File Reference . . . 16
5.2.1 Detailed Description . . . 18
5.2.2 Function Documentation . . . 18
5.2.2.1 gcu_hash_contains() . . . 18
5.2.2.2 gcu_hash_count() . . . 18
5.2.2.3 gcu_hash_create() . . . 18
5.2.2.4 gcu_hash_destroy() . . . 19
5.2.2.5 gcu_hash_get() . . . 19
5.2.2.6 gcu_hash_iterator_get() . . . 20
5.2.2.7 gcu_hash_iterator_next() . . . 20
5.2.2.8 gcu_hash_remove() . . . 21
5.2.2.9 gcu_hash_set() . . . 21

5.3 include/cutil/libver.h File Reference . . . 22
5.3.1 Detailed Description . . . 22
5.3.2 Macro Definition Documentation . . . 23
5.3.2.1 GHOTIIO_CUTIL . . . 23
5.3.2.2 GHOTIIO_CUTIL_CONCAT . . . 23
5.3.2.3 GHOTIIO_CUTIL_CONCAT_INNER . . . 23
5.3.2.4 GHOTIIO_CUTIL_NAME . . . 24
5.4 include/cutil/type.h File Reference . . . 24
5.4.1 Detailed Description . . . 25
5.4.2 Function Documentation . . . 25
5.4.2.1 gcu_type_ui32() . . . 25
5.5 src/debug.c File Reference . . . 26
5.5.1 Function Documentation . . . 27
5.5.1.1 gcu_calloc() . . . 27
5.5.1.2 gcu_free() . . . 27
5.5.1.3 gcu_malloc() . . . 27
5.5.1.4 gcu_realloc() . . . 28
5.6 src/hash.c File Reference . . . 28
5.6.1 Function Documentation . . . 30
5.6.1.1 gcu_hash_contains() . . . 30
5.6.1.2 gcu_hash_count() . . . 30
5.6.1.3 gcu_hash_create() . . . 30
5.6.1.4 gcu_hash_destroy() . . . 31
5.6.1.5 gcu_hash_get() . . . 31
5.6.1.6 gcu_hash_iterator_get() . . . 32
5.6.1.7 gcu_hash_iterator_next() . . . 32
5.6.1.8 gcu_hash_remove() . . . 33
5.6.1.9 gcu_hash_set() . . . 33
5.7 src/type.c File Reference . . . 34
5.7.1 Function Documentation . . . 34
5.7.1.1 gcu_type_ui32() . . . 34
5.8 test/test-debug.cpp File Reference . . . 35
5.8.1 Detailed Description . . . 35
5.9 test/test-hash.cpp File Reference . . . 36
5.9.1 Detailed Description . . . 36
5.10 test/test-type.cpp File Reference . . . 36
5.10.1 Detailed Description . . . 37

Index **39**

Chapter 1

Ghoti.io CUtil Library

1.0.1 Overview

The Ghoti.io CUtil Library is a collection of C libraries to aid in the development of C applications by providing helpful and commonly used tools and features.

1.0.2 Installation

1.0.2.1 Build From Source

```
make build
make install
```

1.0.3 Compiling With The Library

```
cc `pkg-config --libs --cflags ghoti.io-cutil_dev` <YOUR SOURCE FILE>
```

Chapter 2

Class Index

2.1 Class List

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

GCU_Hash_Cell
: Container holding the information for an entry in the hash table 7

GCU_Hash_Iterator
: A container used to hold the state of an iterator which can be used to traverse all elements of a hash table 8

GCU_Hash_Table
: Container holding the information of the hash table 9

GCU_Hash_Value
: Container used to return the result of looking for a hash in the hash table 10

GCU_Type_Union
: A union of all basic types to be used by generic containers 11

Chapter 3

File Index

3.1 File List

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

include/cutil/debug.h
: Header file for debugging-related functions . . . 13

include/cutil/hash.h
: A simple hash table implementation . . . 16

include/cutil/libver.h
: Header file used to control the version numbering and function namespace for all of the library 22

include/cutil/type.h
: Type definitions and utilities for use by the Ghoti.io projects . . . 24

src/debug.c . . . 26

src/hash.c . . . 28

src/type.c . . . 34

test/test-debug.cpp
: Test the behavior of Ghoti.io CUtil debug library and tools . . . 35

test/test-hash.cpp
: Test the behavior of Ghoti.io CUtil hash table library . . . 36

test/test-type.cpp
: Test the behavior of Ghoti.io CUtil type library . . . 36

Chapter 4

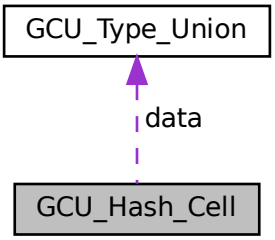
Class Documentation

4.1 GCU_Hash_Cell Struct Reference

Container holding the information for an entry in the hash table.

`#include <hash.h>`

Collaboration diagram for GCU_Hash_Cell:



Public Attributes

- size_t hash

  The hash of the entry.
- GCU_Type_Union data

  The data of the entry.
- bool occupied

  Whether or not the entry has been initialized in some way.
- bool removed

  Whether or not the entry has been removed.

4.1.1 Detailed Description

Container holding the information for an entry in the hash table.

An "entry" is empty (e.g., `occupied = false`) upon creation. By adding and removing entries from the hash table, the `occupied` and `removed` flags will be changed to track the state of each individual cell.

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

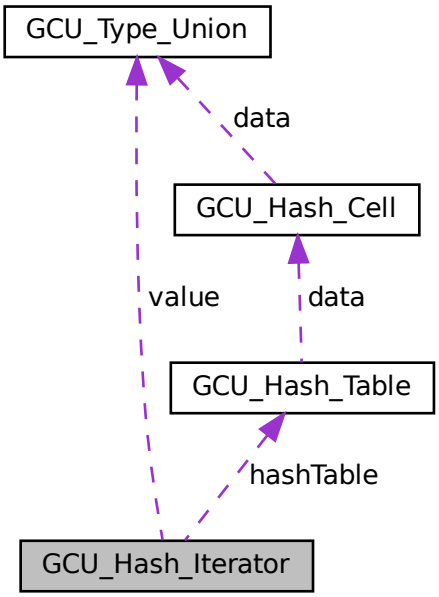
- include/cutil/hash.h

4.2 GCU_Hash_Iterator Struct Reference

A container used to hold the state of an iterator which can be used to traverse all elements of a hash table.

`#include <hash.h>`

Collaboration diagram for GCU_Hash_Iterator:



Public Attributes

- size_t current

  The current index into the hashTable data structure corresponding to the iterator.
- bool exists

  Whether or not the iterator points to valid data.
- GCU_Type_Union value

  The data pointed to by the iterator.
- GCU_Hash_Table * hashTable

  The hash table that the iterator traverses.

4.2.1 Detailed Description

A container used to hold the state of an iterator which can be used to traverse all elements of a hash table.

A hash table may change internal structure upon adding or removing elements, so any such operations may invalidate the behavior of an iterator.

The programmer is responsible to make sure that an iterator is not used improperly after the hash has been modified.

An iterator may contain invalid data, in the case where there is no data through which to iterate. This is indicated by the `exists` field. The programmer is responsible for checking this field before attempting to use the `value` in any way.

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

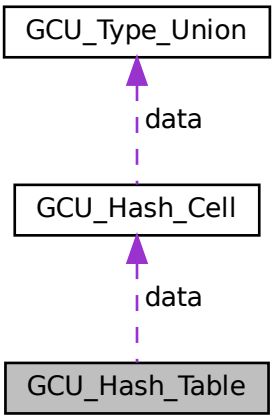
- include/cutil/hash.h

4.3 GCU_Hash_Table Struct Reference

Container holding the information of the hash table.

`#include <hash.h>`

Collaboration diagram for GCU_Hash_Table:



Public Attributes

- size_t capacity

  The total item capacity of the hash table.
- size_t entries

  The count of non-empty cells.
- size_t removed

  The count of non-empty cells that represent elements which have been removed.
- GCU_Hash_Cell * data

  A pointer to the array of data cells.

4.3.1 Detailed Description

Container holding the information of the hash table.

For proper memory management, the programmer is responsible for 4 things:

1. Initialize the hash table using gcu_hash_create().
2. Destroy the has table using gcu_hash_destory().
3. Implementation of any thread-safety synchronization.
4. Life cycle management of the contents of the hash table. The hash table will **not**, for example, attempt to manage any pointers that it may contain upon deletion. The programmer is responsible for all memory management.

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

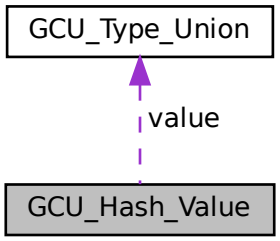
- include/cutil/hash.h

4.4 GCU_Hash_Value Struct Reference

Container used to return the result of looking for a hash in the hash table.

`#include <hash.h>`

Collaboration diagram for GCU_Hash_Value:



Public Attributes

- bool exists

  Whether or not the value exists in the hash table.
- GCU_Type_Union value

  The value found in the table (if it exists).

4.4.1 Detailed Description

Container used to return the result of looking for a hash in the hash table.

Although it may seem strange to return a value as part of a structure, especially when the programmer undoubtedly just wants the value, it is also imperitive that the hash table be able to indicate whether or not the value existed in the table. Both goals are accomplished by this approach.

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

- include/cutil/hash.h

4.5 GCU_Type_Union Union Reference

A union of all basic types to be used by generic containers.

`#include <type.h>`

Public Attributes

- void * **p**
- uint64_t **ui64**
- uint32_t **ui32**
- uint16_t **ui16**
- uint8_t **ui8**
- int64_t **i64**
- int32_t **i32**
- int16_t **i16**
- int8_t **i8**
- char **c**

4.5.1 Detailed Description

A union of all basic types to be used by generic containers.

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

- include/cutil/type.h

Chapter 5

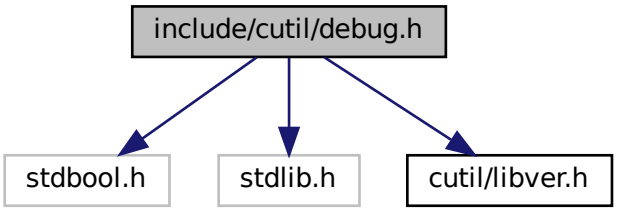
File Documentation

5.1 include/cutil/debug.h File Reference

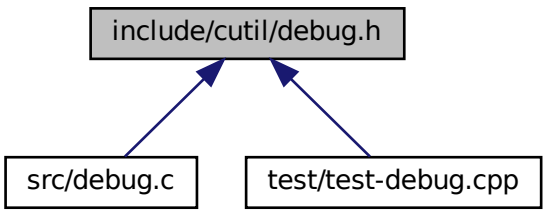
Header file for debugging-related functions.

```
#include <stdbool.h>
#include <stdlib.h>
#include "cutil/libver.h"
```

Include dependency graph for debug.h:



This graph shows which files directly or indirectly include this file:



Functions

- void * gcu_malloc (size_t size, const char *file, size_t line)

  Wrapper for the standard malloc() function.
- void * gcu_calloc (size_t nitems, size_t size, const char *file, size_t line)

  Wrapper for the standard calloc() function.
- void * gcu_realloc (void *pointer, size_t size, const char *file, size_t line)

  Wrapper for the standard realloc() function.
- void gcu_free (void *pointer, const char *file, size_t line)

  Wrapper for the standard free() function.
- void gcu_mem_start (void)

  Signal that intercepted memory management calls should be logged to stderr.
- void gcu_mem_stop (void)

  Signal that intercepted memory management calls should no longer be logged to stderr.

5.1.1 Detailed Description

Header file for debugging-related functions.

Use `#include <cutil/debug.h>` when compiling a file and all calls to `malloc()`, `calloc()`, `realloc()`, and `free()` will be logged to `stderr`. It will only affect code that is compiled with this header.

Logging to `stderr` is enabled by default. It may be disabled by calling gcu_mem_stop(), and re-enabled by calling gcu_mem_start().

You may need to control the logging, but also need to control when the logging starts and stops externally. Obviously, if this header is included, then memory management will also be logged, but this feature can be modified by the use of a `#define` *before* including the header.

By defining `GHOTIIO_CUTIL_DEBUG_DO_NOT_REDECLARE_MEMORY_FUNCTIONS`, the standard `malloc()`, `realloc()`, and `free()` will *not* be redefined, but all other declarations will be intact. In fact, proper compilation of `debug.c` depends on this behavior.

5.1.2 Function Documentation

5.1.2.1 gcu_calloc()

```
void* gcu_calloc (
            size_t nitems,
            size_t size,
            const char * file,
            size_t line )
```

Wrapper for the standard calloc() function.

Parameters

nitems	The number of items to allocate.
size	The number of bytes in each item.
file	The name of the file from which the function was called.
line	The line number on which the function was called.

Returns

The beginning byte of the allocated memory.

5.1.2.2 gcu_free()

```
void gcu_free (
            void * pointer,
            const char * file,
            size_t line )
```

Wrapper for the standard free() function.

Parameters

pointer	The beginning byte of the currently allocated memory.
file	The name of the file from which the function was called.
line	The line number on which the function was called.

5.1.2.3 gcu_malloc()

```
void* gcu_malloc (
            size_t size,
            const char * file,
            size_t line )
```

Wrapper for the standard malloc() function.

Parameters

size	The number of bytes requested.
file	The name of the file from which the function was called.
line	The line number on which the function was called.

Returns

The beginning byte of the allocated memory.

5.1.2.4 gcu_realloc()

```
void* gcu_realloc (
            void * pointer,
```

```
size_t size,
const char * file,
size_t line )
```

Wrapper for the standard realloc() function.

Parameters

pointer	The beginning byte of the currently allocated memory.
size	The newly requested size.
file	The name of the file from which the function was called.
line	The line number on which the function was called.

Returns

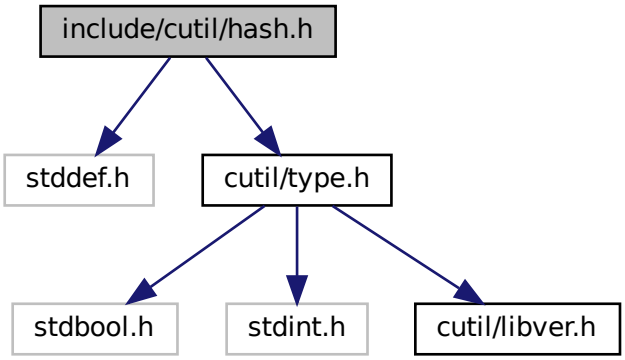
The beginning byte of the reallocated memory.

5.2 include/cutil/hash.h File Reference

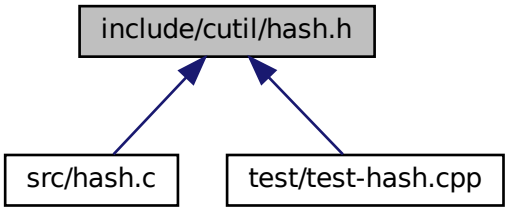
A simple hash table implementation.

```
#include <stddef.h>
#include "cutil/type.h"
```

Include dependency graph for hash.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct GCU_Hash_Value

  Container used to return the result of looking for a hash in the hash table.
- struct GCU_Hash_Cell

  Container holding the information for an entry in the hash table.
- struct GCU_Hash_Table

  Container holding the information of the hash table.
- struct GCU_Hash_Iterator

  A container used to hold the state of an iterator which can be used to traverse all elements of a hash table.

Functions

- GCU_Hash_Table * gcu_hash_create (size_t count)

  Create a hash table structure.
- void gcu_hash_destroy (GCU_Hash_Table *hashTable)

  Destroy a hash table structure and clean up memory allocations.
- bool gcu_hash_set (GCU_Hash_Table *hashTable, size_t hash, GCU_Type_Union value)

  Set a value in the hash table.
- GCU_Hash_Value gcu_hash_get (GCU_Hash_Table *hashTable, size_t hash)

  Get a value from the hash table (if it exists).
- bool gcu_hash_contains (GCU_Hash_Table *hashTable, size_t hash)

  Check to see whether or not a hash table contains a specific hash.
- bool gcu_hash_remove (GCU_Hash_Table *hashTable, size_t hash)

  Remove a hash from the table.
- size_t gcu_hash_count (GCU_Hash_Table *hashTable)

  Get a count of active entries in the hash table.
- GCU_Hash_Iterator gcu_hash_iterator_get (GCU_Hash_Table *hashTable)

  Get an iterator which can be used to iterate through the entries of the hash table.
- GCU_Hash_Iterator gcu_hash_iterator_next (GCU_Hash_Iterator iterator)

  Get an iterator to the next element in the table (if it exists).

5.2.1 Detailed Description

A simple hash table implementation.

5.2.2 Function Documentation

5.2.2.1 gcu_hash_contains()

```
bool gcu_hash_contains (
            GCU_Hash_Table * hashTable,
            size_t hash )
```

Check to see whether or not a hash table contains a specific hash.

Parameters

hashTable	The hash table structure on which to operate.
hash	The hash whose associated value will be searched for.

Returns

`true` if the hash is in the table, `false` otherwise.

5.2.2.2 gcu_hash_count()

```
size_t gcu_hash_count (
            GCU_Hash_Table * hashTable )
```

Get a count of active entries in the hash table.

Parameters

hashTable	The hash table structure on which to operate.

Returns

The count of active entries in the hash table.

5.2.2.3 gcu_hash_create()

```
GCU_Hash_Table* gcu_hash_create (
            size_t count )
```

Create a hash table structure.

All invocations of a hash table must have a corresponding gcu_hash_destroy() call in order to clean up dynamically-allocated memory.

The hash table will manage the final size of container's memory based on the number of elements that have been added. The container's memory will be expanded automatically when needed to accomodate new insertions, which can cause an unexpected delay. Such rebuilding costs can be avoided by proper setting of the `count` variable during creation of the hash table.

Parameters

count	The number of items anticipated to be stored in the hash table.

Returns

A struct containing the hash table information.

5.2.2.4 gcu_hash_destroy()

```
void gcu_hash_destroy (
            GCU_Hash_Table * hashTable )
```

Destroy a hash table structure and clean up memory allocations.

This function will not address any memory allocations of the elements themselves (if any). The programmer is responsible for controlling any memory management on behalf of the elements.

Parameters

hashTable	The hash table structure to be destroyed.

5.2.2.5 gcu_hash_get()

```
GCU_Hash_Value gcu_hash_get (
            GCU_Hash_Table * hashTable,
            size_t hash )
```

Get a value from the hash table (if it exists).

Parameters

hashTable	The hash table structure on which to operate.
hash	The hash whose associated value will be searched for.

Returns

A result that indicates the success or failure of the operation, as well as the associated value (if it exists).

5.2.2.6 gcu_hash_iterator_get()

```
GCU_Hash_Iterator gcu_hash_iterator_get (
            GCU_Hash_Table * hashTable )
```

Get an iterator which can be used to iterate through the entries of the hash table.

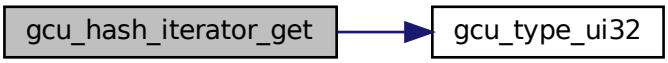
Parameters

hashTable	The hash table structure on which to operate.

Returns

An iterator pointing to the first element in the hash table (if it exists).

Here is the call graph for this function:



5.2.2.7 gcu_hash_iterator_next()

```
GCU_Hash_Iterator gcu_hash_iterator_next (
            GCU_Hash_Iterator iterator )
```

Get an iterator to the next element in the table (if it exists).

Any change to the hash table (such as setting a value) might alter the underlying structure of the hash table, which would invalidate the iterator. Any call to gcu_hash_set(), therefore, should be considered as an invalidation of any iterators associated with the hash table.

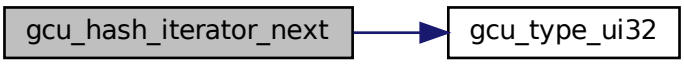
Parameters

iterator	The iterator from which to calculate and return the next iterator.

Returns

An iterator pointing to the next element in the table (if it exists).

Here is the call graph for this function:



5.2.2.8 gcu_hash_remove()

```
bool gcu_hash_remove (
            GCU_Hash_Table * hashTable,
            size_t hash )
```

Remove a hash from the table.

The hash table does not manage the values in the table. Therefore, if an entry is removed from the hash table, then it is up to the programmer to perform any additional work (such as memory cleanup of the value).

Parameters

hashTable	The hash table structure on which to operate.
hash	The hash whose associated value will be removed from the table.

Returns

`true` if the entry existed and was removed, `false` otherwise.

5.2.2.9 gcu_hash_set()

```
bool gcu_hash_set (
            GCU_Hash_Table * hashTable,
            size_t hash,
            GCU_Type_Union value )
```

Set a value in the hash table.

Setting a value may trigger a resize of the hash table. This can be avoided entirely by setting an appropriate `count` value when creating the hash table with gcu_hash_create().

Parameters

hashTable	The hash table structure on which to operate.
hash	The hash associated with the value.
value	The value to insert into the hash table.

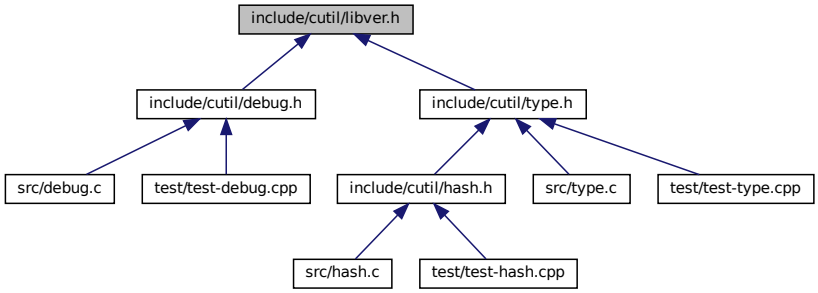
Returns

`true` on success, `false` on failure.

5.3 include/cutil/libver.h File Reference

Header file used to control the version numbering and function namespace for all of the library.

This graph shows which files directly or indirectly include this file:



Macros

- #define GHOTIIO_CUTIL_NAME ghotiio_cutil_dev

  Used in conjunction with the GHOTIIO_CUTIL...
- #define GHOTIIO_CUTIL_VERSION "dev"

  String representation of the version, provided as a convenience to the programmer.
- #define GHOTIIO_CUTIL(NAME) GHOTIIO_CUTIL_CONCAT(GHOTIIO_CUTIL_NAME, _ ## NAME)

  Macro to generate a "namespaced" version of an identifier.
- #define GHOTIIO_CUTIL_CONCAT_INNER(a, b) a ## b

  Helper macro to concatenate the `#define`s properly.
- #define GHOTIIO_CUTIL_CONCAT(a, b) GHOTIIO_CUTIL_CONCAT_INNER(a,b)

  Helper macro to concatenate the `#define`s properly.

5.3.1 Detailed Description

Header file used to control the version numbering and function namespace for all of the library.

5.3.2 Macro Definition Documentation

5.3.2.1 GHOTIIO_CUTIL

```
#define GHOTIIO_CUTIL(
              NAME ) GHOTIIO_CUTIL_CONCAT(GHOTIIO_CUTIL_NAME, _ ## NAME)
```

Macro to generate a "namespaced" version of an identifier.

Parameters

NAME	The name which will be prepended with the GHOTIIO_CUTIL_NAME.

5.3.2.2 GHOTIIO_CUTIL_CONCAT

```
#define GHOTIIO_CUTIL_CONCAT(
              a,
              b ) GHOTIIO_CUTIL_CONCAT_INNER(a,b)
```

Helper macro to concatenate the #defines properly.

It requires two levels of processing.

Parameters

a	The first part of the identifier.
b	The second part of the identifier.

Returns

A call to the GHOTIIO_CUTIL_CONCAT_INNER() macro.

5.3.2.3 GHOTIIO_CUTIL_CONCAT_INNER

```
#define GHOTIIO_CUTIL_CONCAT_INNER(
              a,
              b ) a ## b
```

Helper macro to concatenate the #defines properly.

It requires two levels of processing.

This macro should only be called by the GHOTIIO_CUTIL_CONCAT() macro.

Parameters

a	The first part of the identifier.
b	The second part of the identifier.

Returns

The concatenation of `a` to `b`.

5.3.2.4 GHOTIIO_CUTIL_NAME

```
#define GHOTIIO_CUTIL_NAME ghotiio_cutil_dev
```

Used in conjunction with the GHOTIIO_CUTIL...

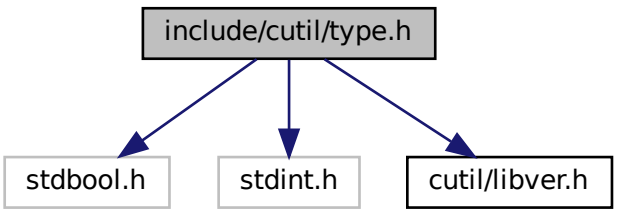
macros to produce a namespaced function name for use by all exported functions in this library.

5.4 include/cutil/type.h File Reference

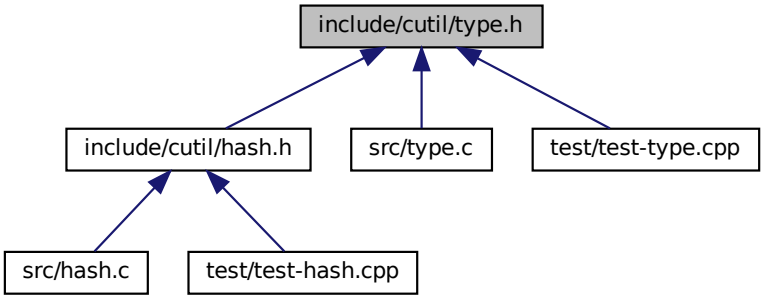
Type definitions and utilities for use by the Ghoti.io projects.

```
#include <stdbool.h>
#include <stdint.h>
#include "cutil/libver.h"
```

Include dependency graph for type.h:



This graph shows which files directly or indirectly include this file:



Classes

- union GCU_Type_Union

  A union of all basic types to be used by generic containers.

Functions

- GCU_Type_Union gcu_type_ui32 (uint32_t val)

  Create a union variable with the type uint32_t.

5.4.1 Detailed Description

Type definitions and utilities for use by the Ghoti.io projects.

5.4.2 Function Documentation

5.4.2.1 gcu_type_ui32()

```
GCU_Type_Union gcu_type_ui32 (
            uint32_t val )
```

Create a union variable with the type uint32_t.

Parameters

val	The value to put into the union.

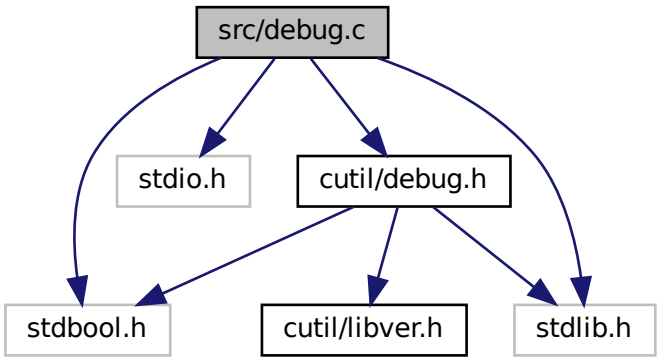
Returns

The union variable.

5.5 src/debug.c File Reference

```
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include "cutil/debug.h"
```

Include dependency graph for debug.c:



Functions

- void * gcu_malloc (size_t size, const char *file, size_t line)

  Wrapper for the standard malloc() function.
- void * gcu_calloc (size_t nitems, size_t size, const char *file, size_t line)

  Wrapper for the standard calloc() function.
- void * gcu_realloc (void *pointer, size_t size, const char *file, size_t line)

  Wrapper for the standard realloc() function.
- void gcu_free (void *pointer, const char *file, size_t line)

  Wrapper for the standard free() function.
- void gcu_mem_start (void)

  Signal that intercepted memory management calls should be logged to stderr.
- void gcu_mem_stop (void)

  Signal that intercepted memory management calls should no longer be logged to stderr.

Variables

- bool **capture** = true

5.5.1 Function Documentation

5.5.1.1 gcu_calloc()

```
void* gcu_calloc (
            size_t nitems,
            size_t size,
            const char * file,
            size_t line )
```

Wrapper for the standard calloc() function.

Parameters

nitems	The number of items to allocate.
size	The number of bytes in each item.
file	The name of the file from which the function was called.
line	The line number on which the function was called.

Returns

The beginning byte of the allocated memory.

5.5.1.2 gcu_free()

```
void gcu_free (
            void * pointer,
            const char * file,
            size_t line )
```

Wrapper for the standard free() function.

Parameters

pointer	The beginning byte of the currently allocated memory.
file	The name of the file from which the function was called.
line	The line number on which the function was called.

5.5.1.3 gcu_malloc()

```
void* gcu_malloc (
            size_t size,
```

```
            const char * file,
            size_t line )
```

Wrapper for the standard malloc() function.

Parameters

size	The number of bytes requested.
file	The name of the file from which the function was called.
line	The line number on which the function was called.

Returns

The beginning byte of the allocated memory.

5.5.1.4 gcu_realloc()

```
void* gcu_realloc (
            void * pointer,
            size_t size,
            const char * file,
            size_t line )
```

Wrapper for the standard realloc() function.

Parameters

pointer	The beginning byte of the currently allocated memory.
size	The newly requested size.
file	The name of the file from which the function was called.
line	The line number on which the function was called.

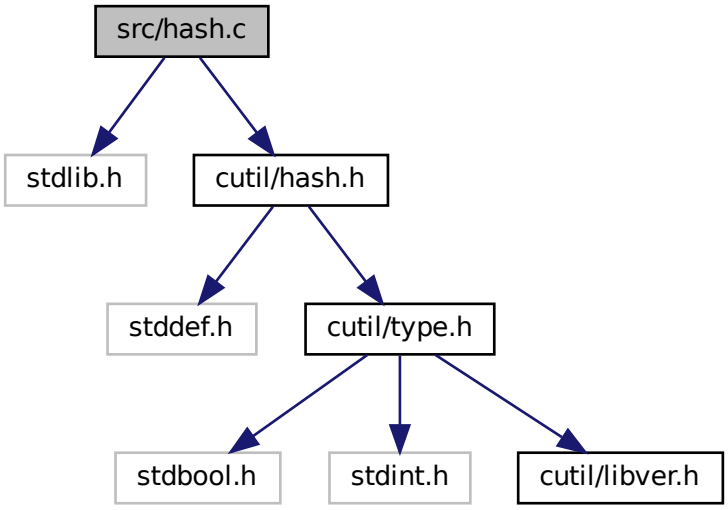
Returns

The beginning byte of the reallocated memory.

5.6 src/hash.c File Reference

```
#include <stdlib.h>
#include "cutil/hash.h"
```

Include dependency graph for hash.c:



Macros

- #define **GROWTH_FACTOR** 1.25

Functions

- GCU_Hash_Table * gcu_hash_create (size_t count)

  Create a hash table structure.
- void gcu_hash_destroy (GCU_Hash_Table *hashTable)

  Destroy a hash table structure and clean up memory allocations.
- bool gcu_hash_set (GCU_Hash_Table *hashTable, size_t hash, GCU_Type_Union value)

  Set a value in the hash table.
- GCU_Hash_Value gcu_hash_get (GCU_Hash_Table *hashTable, size_t hash)

  Get a value from the hash table (if it exists).
- bool gcu_hash_contains (GCU_Hash_Table *hashTable, size_t hash)

  Check to see whether or not a hash table contains a specific hash.
- bool gcu_hash_remove (GCU_Hash_Table *hashTable, size_t hash)

  Remove a hash from the table.
- size_t gcu_hash_count (GCU_Hash_Table *hashTable)

  Get a count of active entries in the hash table.
- GCU_Hash_Iterator gcu_hash_iterator_get (GCU_Hash_Table *hashTable)

  Get an iterator which can be used to iterate through the entries of the hash table.
- GCU_Hash_Iterator gcu_hash_iterator_next (GCU_Hash_Iterator iterator)

  Get an iterator to the next element in the table (if it exists).

5.6.1 Function Documentation

5.6.1.1 gcu_hash_contains()

```
bool gcu_hash_contains (
            GCU_Hash_Table * hashTable,
            size_t hash )
```

Check to see whether or not a hash table contains a specific hash.

Parameters

hashTable	The hash table structure on which to operate.
hash	The hash whose associated value will be searched for.

Returns

`true` if the hash is in the table, `false` otherwise.

5.6.1.2 gcu_hash_count()

```
size_t gcu_hash_count (
            GCU_Hash_Table * hashTable )
```

Get a count of active entries in the hash table.

Parameters

hashTable	The hash table structure on which to operate.

Returns

The count of active entries in the hash table.

5.6.1.3 gcu_hash_create()

```
GCU_Hash_Table* gcu_hash_create (
            size_t count )
```

Create a hash table structure.

All invocations of a hash table must have a corresponding gcu_hash_destroy() call in order to clean up dynamically-allocated memory.

The hash table will manage the final size of container's memory based on the number of elements that have been added. The container's memory will be expanded automatically when needed to accomodate new insertions, which can cause an unexpected delay. Such rebuilding costs can be avoided by proper setting of the `count` variable during creation of the hash table.

Parameters

count	The number of items anticipated to be stored in the hash table.

Returns

A struct containing the hash table information.

5.6.1.4 gcu_hash_destroy()

```
void gcu_hash_destroy (
            GCU_Hash_Table * hashTable )
```

Destroy a hash table structure and clean up memory allocations.

This function will not address any memory allocations of the elements themselves (if any). The programmer is responsible for controlling any memory management on behalf of the elements.

Parameters

hashTable	The hash table structure to be destroyed.

5.6.1.5 gcu_hash_get()

```
GCU_Hash_Value gcu_hash_get (
            GCU_Hash_Table * hashTable,
            size_t hash )
```

Get a value from the hash table (if it exists).

Parameters

hashTable	The hash table structure on which to operate.
hash	The hash whose associated value will be searched for.

Returns

A result that indicates the success or failure of the operation, as well as the associated value (if it exists).

5.6.1.6 gcu_hash_iterator_get()

```
GCU_Hash_Iterator gcu_hash_iterator_get (
            GCU_Hash_Table * hashTable )
```

Get an iterator which can be used to iterate through the entries of the hash table.

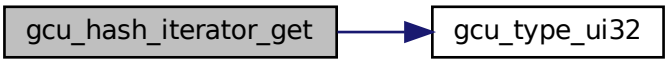
Parameters

hashTable	The hash table structure on which to operate.

Returns

An iterator pointing to the first element in the hash table (if it exists).

Here is the call graph for this function:



5.6.1.7 gcu_hash_iterator_next()

```
GCU_Hash_Iterator gcu_hash_iterator_next (
            GCU_Hash_Iterator iterator )
```

Get an iterator to the next element in the table (if it exists).

Any change to the hash table (such as setting a value) might alter the underlying structure of the hash table, which would invalidate the iterator. Any call to gcu_hash_set(), therefore, should be considered as an invalidation of any iterators associated with the hash table.

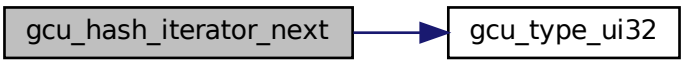
Parameters

iterator	The iterator from which to calculate and return the next iterator.

Returns

An iterator pointing to the next element in the table (if it exists).

Here is the call graph for this function:



5.6.1.8 gcu_hash_remove()

```
bool gcu_hash_remove (
            GCU_Hash_Table * hashTable,
            size_t hash )
```

Remove a hash from the table.

The hash table does not manage the values in the table. Therefore, if an entry is removed from the hash table, then it is up to the programmer to perform any additional work (such as memory cleanup of the value).

Parameters

hashTable	The hash table structure on which to operate.
hash	The hash whose associated value will be removed from the table.

Returns

`true` if the entry existed and was removed, `false` otherwise.

5.6.1.9 gcu_hash_set()

```
bool gcu_hash_set (
            GCU_Hash_Table * hashTable,
            size_t hash,
            GCU_Type_Union value )
```

Set a value in the hash table.

Setting a value may trigger a resize of the hash table. This can be avoided entirely by setting an appropriate `count` value when creating the hash table with gcu_hash_create().

Parameters

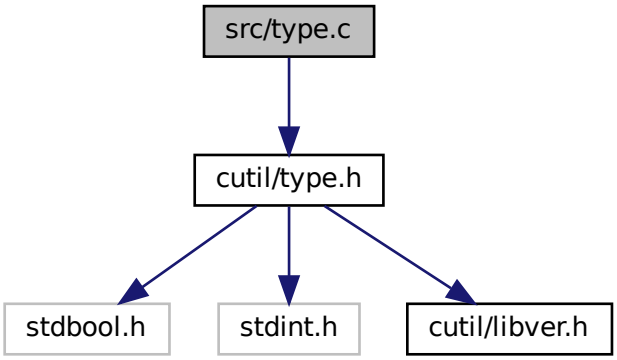
hashTable	The hash table structure on which to operate.
hash	The hash associated with the value.
value	The value to insert into the hash table.

Returns

`true` on success, `false` on failure.

5.7 src/type.c File Reference

`#include "cutil/type.h"`
Include dependency graph for type.c:



Functions

- GCU_Type_Union gcu_type_ui32 (uint32_t val)
  Create a union variable with the type uint32_t.

5.7.1 Function Documentation

5.7.1.1 gcu_type_ui32()

```
GCU_Type_Union gcu_type_ui32 (
            uint32_t val )
```

Create a union variable with the type uint32_t.

Parameters

val	The value to put into the union.

Returns

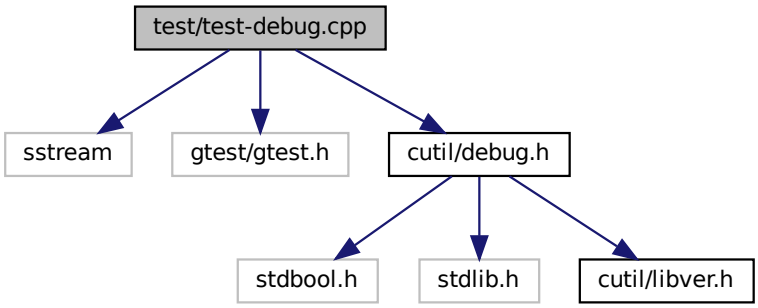
The union variable.

5.8 test/test-debug.cpp File Reference

Test the behavior of Ghoti.io CUtil debug library and tools.

```
#include <sstream>
#include <gtest/gtest.h>
#include "cutil/debug.h"
```

Include dependency graph for test-debug.cpp:



Functions

- **TEST** (Memory, MallocReallocFree)
- **TEST** (Memory, CallocFree)
- **TEST** (Memory, StopStartCapture)
- int **main** (int argc, char ∗∗argv)

5.8.1 Detailed Description

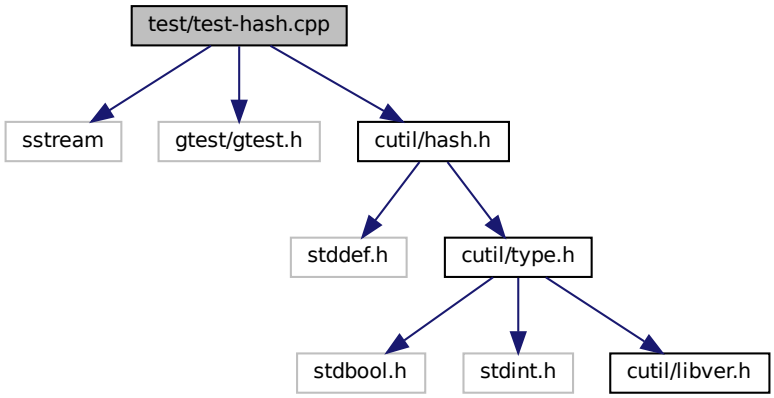
Test the behavior of Ghoti.io CUtil debug library and tools.

5.9 test/test-hash.cpp File Reference

Test the behavior of Ghoti.io CUtil hash table library.

```
#include <sstream>
#include <gtest/gtest.h>
#include "cutil/hash.h"
```

Include dependency graph for test-hash.cpp:



Functions

- **TEST** (Hash, CreateEmpty)
- **TEST** (Hash, Create)
- **TEST** (Hash, Set)
- **TEST** (Hash, Remove)
- **TEST** (Hash, IteratorOnEmpty)
- **TEST** (Hash, Iterator)
- int **main** (int argc, char ∗∗argv)

5.9.1 Detailed Description

Test the behavior of Ghoti.io CUtil hash table library.

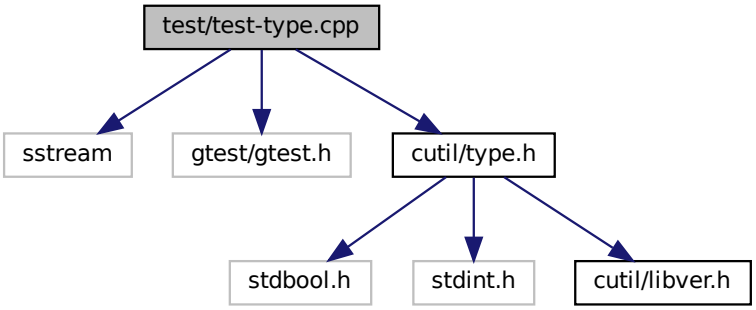
5.10 test/test-type.cpp File Reference

Test the behavior of Ghoti.io CUtil type library.

```
#include <sstream>
#include <gtest/gtest.h>
```

```
#include "cutil/type.h"
```

Include dependency graph for test-type.cpp:



Functions

- int **main** (int argc, char ∗∗argv)

5.10.1 Detailed Description

Test the behavior of Ghoti.io CUtil type library.

Index

debug.c
    gcu_calloc, 27
    gcu_free, 27
    gcu_malloc, 27
    gcu_realloc, 28
debug.h
    gcu_calloc, 14
    gcu_free, 15
    gcu_malloc, 15
    gcu_realloc, 15

gcu_calloc
    debug.c, 27
    debug.h, 14
gcu_free
    debug.c, 27
    debug.h, 15
GCU_Hash_Cell, 7
gcu_hash_contains
    hash.c, 30
    hash.h, 18
gcu_hash_count
    hash.c, 30
    hash.h, 18
gcu_hash_create
    hash.c, 30
    hash.h, 18
gcu_hash_destroy
    hash.c, 31
    hash.h, 19
gcu_hash_get
    hash.c, 31
    hash.h, 19
GCU_Hash_Iterator, 8
gcu_hash_iterator_get
    hash.c, 32
    hash.h, 20
gcu_hash_iterator_next
    hash.c, 32
    hash.h, 20
gcu_hash_remove
    hash.c, 33
    hash.h, 21
gcu_hash_set
    hash.c, 33
    hash.h, 21
GCU_Hash_Table, 9
GCU_Hash_Value, 10
gcu_malloc
    debug.c, 27
    debug.h, 15
gcu_realloc
    debug.c, 28
    debug.h, 15
gcu_type_ui32
    type.c, 34
    type.h, 25
GCU_Type_Union, 11
GHOTIIO_CUTIL
    libver.h, 23
GHOTIIO_CUTIL_CONCAT
    libver.h, 23
GHOTIIO_CUTIL_CONCAT_INNER
    libver.h, 23
GHOTIIO_CUTIL_NAME
    libver.h, 24

hash.c
    gcu_hash_contains, 30
    gcu_hash_count, 30
    gcu_hash_create, 30
    gcu_hash_destroy, 31
    gcu_hash_get, 31
    gcu_hash_iterator_get, 32
    gcu_hash_iterator_next, 32
    gcu_hash_remove, 33
    gcu_hash_set, 33
hash.h
    gcu_hash_contains, 18
    gcu_hash_count, 18
    gcu_hash_create, 18
    gcu_hash_destroy, 19
    gcu_hash_get, 19
    gcu_hash_iterator_get, 20
    gcu_hash_iterator_next, 20
    gcu_hash_remove, 21
    gcu_hash_set, 21

include/cutil/debug.h, 13
include/cutil/hash.h, 16
include/cutil/libver.h, 22
include/cutil/type.h, 24

libver.h
    GHOTIIO_CUTIL, 23
    GHOTIIO_CUTIL_CONCAT, 23
    GHOTIIO_CUTIL_CONCAT_INNER, 23
    GHOTIIO_CUTIL_NAME, 24

src/debug.c, 26

src/hash.c, 28
src/type.c, 34

test/test-debug.cpp, 35
test/test-hash.cpp, 36
test/test-type.cpp, 36
type.c
    gcu_type_ui32, 34
type.h
    gcu_type_ui32, 25