

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_Hash16_Cell Struct Reference	7
4.1.1 Detailed Description	8
4.2 GCU_Hash16_Iterator Struct Reference	8
4.2.1 Detailed Description	9
4.3 GCU_Hash16_Table Struct Reference	9
4.3.1 Detailed Description	10
4.4 GCU_Hash16_Value Struct Reference	10
4.4.1 Detailed Description	11
4.5 GCU_Hash32_Cell Struct Reference	11
4.5.1 Detailed Description	12
4.6 GCU_Hash32_Iterator Struct Reference	12
4.6.1 Detailed Description	13
4.7 GCU_Hash32_Table Struct Reference	13
4.7.1 Detailed Description	14
4.8 GCU_Hash32_Value Struct Reference	14
4.8.1 Detailed Description	15
4.9 GCU_Hash64_Cell Struct Reference	15
4.9.1 Detailed Description	16
4.10 GCU_Hash64_Iterator Struct Reference	16
4.10.1 Detailed Description	17
4.11 GCU_Hash64_Table Struct Reference	17
4.11.1 Detailed Description	18
4.12 GCU_Hash64_Value Struct Reference	18
4.12.1 Detailed Description	19
4.13 GCU_Hash8_Cell Struct Reference	19
4.13.1 Detailed Description	20
4.14 GCU_Hash8_Iterator Struct Reference	20
4.14.1 Detailed Description	21
4.15 GCU_Hash8_Table Struct Reference	21
4.15.1 Detailed Description	22

4.16 GCU_Hash8_Value Struct Reference	22
4.16.1 Detailed Description	23
4.17 GCU_Type16_Union Union Reference	23
4.17.1 Detailed Description	23
4.18 GCU_Type32_Union Union Reference	23
4.18.1 Detailed Description	24
4.19 GCU_Type64_Union Union Reference	24
4.19.1 Detailed Description	24
4.20 GCU_Type8_Union Union Reference	24
4.20.1 Detailed Description	25
4.21 GCU_Vector16 Struct Reference	25
4.21.1 Detailed Description	25
4.22 GCU_Vector32 Struct Reference	26
4.22.1 Detailed Description	26
4.23 GCU_Vector64 Struct Reference	27
4.23.1 Detailed Description	27
4.24 GCU_Vector8 Struct Reference	28
4.24.1 Detailed Description	28
5 File Documentation	29
5.1 include/cutil/debug.h File Reference	29
5.1.1 Detailed Description	30
5.1.2 Function Documentation	30
5.1.2.1 gcu_calloc()	30
5.1.2.2 gcu_free()	31
5.1.2.3 gcu_malloc()	31
5.1.2.4 gcu_realloc()	31
5.2 include/cutil/float.h File Reference	32
5.2.1 Detailed Description	33
5.3 include/cutil/hash.h File Reference	33
5.3.1 Detailed Description	36
5.3.2 Function Documentation	36
5.3.2.1 gcu_hash16_contains()	36
5.3.2.2 gcu_hash16_count()	36
5.3.2.3 gcu_hash16_create()	37
5.3.2.4 gcu_hash16_destroy()	37
5.3.2.5 gcu_hash16_get()	37
5.3.2.6 gcu_hash16_iterator_get()	38
5.3.2.7 gcu_hash16_iterator_next()	38
5.3.2.8 gcu_hash16_remove()	39
5.3.2.9 gcu_hash16_set()	40
5.3.2.10 gcu_hash32_contains()	40

5.3.2.11 gcu_hash32_count()	40
5.3.2.12 gcu_hash32_create()	41
5.3.2.13 gcu_hash32_destroy()	41
5.3.2.14 gcu_hash32_get()	42
5.3.2.15 gcu_hash32_iterator_get()	42
5.3.2.16 gcu_hash32_iterator_next()	43
5.3.2.17 gcu_hash32_remove()	43
5.3.2.18 gcu_hash32_set()	44
5.3.2.19 gcu_hash64_contains()	44
5.3.2.20 gcu_hash64_count()	45
5.3.2.21 gcu_hash64_create()	45
5.3.2.22 gcu_hash64_destroy()	45
5.3.2.23 gcu_hash64_get()	46
5.3.2.24 gcu_hash64_iterator_get()	46
5.3.2.25 gcu_hash64_iterator_next()	47
5.3.2.26 gcu_hash64_remove()	48
5.3.2.27 gcu_hash64_set()	48
5.3.2.28 gcu_hash8_contains()	48
5.3.2.29 gcu_hash8_count()	50
5.3.2.30 gcu_hash8_create()	50
5.3.2.31 gcu_hash8_destroy()	51
5.3.2.32 gcu_hash8_get()	51
5.3.2.33 gcu_hash8_iterator_get()	51
5.3.2.34 gcu_hash8_iterator_next()	52
5.3.2.35 gcu_hash8_remove()	53
5.3.2.36 gcu_hash8_set()	53
5.4 include/cutil/libver.h File Reference	53
5.4.1 Detailed Description	54
5.4.2 Macro Definition Documentation	54
5.4.2.1 GHOTIIO_CUTIL	54
5.4.2.2 GHOTIIO_CUTIL_CONCAT	55
5.4.2.3 GHOTIIO_CUTIL_CONCAT_INNER	55
5.4.2.4 GHOTIIO_CUTIL_NAME	56
5.5 include/cutil/type.h File Reference	56
5.5.1 Detailed Description	58
5.5.2 Function Documentation	58
5.5.2.1 gcu_type16_c()	58
5.5.2.2 gcu_type16_i16()	59
5.5.2.3 gcu_type16_i8()	59
5.5.2.4 gcu_type16_ui16()	59
5.5.2.5 gcu_type16_ui8()	60
5.5.2.6 gcu_type32_c()	60

5.5.2.7 gcu_type32_f32()	60
5.5.2.8 gcu_type32_i16()	61
5.5.2.9 gcu_type32_i32()	61
5.5.2.10 gcu_type32_i8()	61
5.5.2.11 gcu_type32_ui16()	62
5.5.2.12 gcu_type32_ui32()	62
5.5.2.13 gcu_type32_ui64()	62
5.5.2.14 gcu_type32_ui8()	64
5.5.2.15 gcu_type64_c()	64
5.5.2.16 gcu_type64_f32()	64
5.5.2.17 gcu_type64_f64()	66
5.5.2.18 gcu_type64_i16()	66
5.5.2.19 gcu_type64_i32()	66
5.5.2.20 gcu_type64_i64()	68
5.5.2.21 gcu_type64_i8()	68
5.5.2.22 gcu_type64_p()	68
5.5.2.23 gcu_type64_ui16()	70
5.5.2.24 gcu_type64_ui32()	70
5.5.2.25 gcu_type64_ui64()	70
5.5.2.26 gcu_type64_ui8()	72
5.5.2.27 gcu_type8_c()	72
5.5.2.28 gcu_type8_i8()	72
5.5.2.29 gcu_type8_ui8()	74
5.6 include/cutil/vector.h File Reference	74
5.6.1 Detailed Description	76
5.6.2 Function Documentation	76
5.6.2.1 gcu_vector16_append()	76
5.6.2.2 gcu_vector16_count()	77
5.6.2.3 gcu_vector16_create()	77
5.6.2.4 gcu_vector16_destroy()	78
5.6.2.5 gcu_vector32_append()	78
5.6.2.6 gcu_vector32_count()	78
5.6.2.7 gcu_vector32_create()	79
5.6.2.8 gcu_vector32_destroy()	79
5.6.2.9 gcu_vector64_append()	80
5.6.2.10 gcu_vector64_count()	80
5.6.2.11 gcu_vector64_create()	80
5.6.2.12 gcu_vector64_destroy()	82
5.6.2.13 gcu_vector8_append()	82
5.6.2.14 gcu_vector8_count()	83
5.6.2.15 gcu_vector8_create()	83
5.6.2.16 gcu_vector8_destroy()	83

5.7 src/debug.c File Reference	84
5.7.1 Function Documentation	85
5.7.1.1 gcu_calloc()	85
5.7.1.2 gcu_free()	85
5.7.1.3 gcu_malloc()	85
5.7.1.4 gcu_realloc()	86
5.8 src/float_identifier.c File Reference	86
5.8.1 Detailed Description	87
5.9 src/hash.c File Reference	87
5.9.1 Function Documentation	90
5.9.1.1 gcu_hash16_contains()	90
5.9.1.2 gcu_hash16_count()	90
5.9.1.3 gcu_hash16_create()	91
5.9.1.4 gcu_hash16_destroy()	91
5.9.1.5 gcu_hash16_get()	91
5.9.1.6 gcu_hash16_iterator_get()	92
5.9.1.7 gcu_hash16_iterator_next()	92
5.9.1.8 gcu_hash16_remove()	93
5.9.1.9 gcu_hash16_set()	93
5.9.1.10 gcu_hash32_contains()	94
5.9.1.11 gcu_hash32_count()	94
5.9.1.12 gcu_hash32_create()	95
5.9.1.13 gcu_hash32_destroy()	95
5.9.1.14 gcu_hash32_get()	95
5.9.1.15 gcu_hash32_iterator_get()	96
5.9.1.16 gcu_hash32_iterator_next()	96
5.9.1.17 gcu_hash32_remove()	97
5.9.1.18 gcu_hash32_set()	98
5.9.1.19 gcu_hash64_contains()	98
5.9.1.20 gcu_hash64_count()	98
5.9.1.21 gcu_hash64_create()	99
5.9.1.22 gcu_hash64_destroy()	99
5.9.1.23 gcu_hash64_get()	100
5.9.1.24 gcu_hash64_iterator_get()	100
5.9.1.25 gcu_hash64_iterator_next()	101
5.9.1.26 gcu_hash64_remove()	101
5.9.1.27 gcu_hash64_set()	102
5.9.1.28 gcu_hash8_contains()	102
5.9.1.29 gcu_hash8_count()	103
5.9.1.30 gcu_hash8_create()	103
5.9.1.31 gcu_hash8_destroy()	103
5.9.1.32 gcu_hash8_get()	104

5.9.1.33 gcu_hash8_iterator_get()	104
5.9.1.34 gcu_hash8_iterator_next()	105
5.9.1.35 gcu_hash8_remove()	106
5.9.1.36 gcu_hash8_set()	106
5.10 src/type.c File Reference	106
5.10.1 Function Documentation	108
5.10.1.1 gcu_type16_c()	108
5.10.1.2 gcu_type16_i16()	109
5.10.1.3 gcu_type16_i8()	109
5.10.1.4 gcu_type16_ui16()	109
5.10.1.5 gcu_type16_ui8()	110
5.10.1.6 gcu_type32_c()	110
5.10.1.7 gcu_type32_f32()	110
5.10.1.8 gcu_type32_i16()	112
5.10.1.9 gcu_type32_i32()	112
5.10.1.10 gcu_type32_i8()	112
5.10.1.11 gcu_type32_ui16()	114
5.10.1.12 gcu_type32_ui32()	114
5.10.1.13 gcu_type32_ui8()	114
5.10.1.14 gcu_type64_c()	116
5.10.1.15 gcu_type64_f32()	116
5.10.1.16 gcu_type64_f64()	116
5.10.1.17 gcu_type64_i16()	118
5.10.1.18 gcu_type64_i32()	118
5.10.1.19 gcu_type64_i64()	118
5.10.1.20 gcu_type64_i8()	120
5.10.1.21 gcu_type64_p()	120
5.10.1.22 gcu_type64_ui16()	120
5.10.1.23 gcu_type64_ui32()	122
5.10.1.24 gcu_type64_ui64()	122
5.10.1.25 gcu_type64_ui8()	122
5.10.1.26 gcu_type8_c()	124
5.10.1.27 gcu_type8_i8()	124
5.10.1.28 gcu_type8_ui8()	124
5.11 src/vector.c File Reference	126
5.11.1 Function Documentation	127
5.11.1.1 gcu_vector16_append()	127
5.11.1.2 gcu_vector16_count()	128
5.11.1.3 gcu_vector16_create()	128
5.11.1.4 gcu_vector16_destroy()	128
5.11.1.5 gcu_vector32_append()	130
5.11.1.6 gcu_vector32_count()	130

5.11.1.7 gcu_vector32_create()	130
5.11.1.8 gcu_vector32_destroy()	131
5.11.1.9 gcu_vector64_append()	131
5.11.1.10 gcu_vector64_count()	132
5.11.1.11 gcu_vector64_create()	132
5.11.1.12 gcu_vector64_destroy()	133
5.11.1.13 gcu_vector8_append()	133
5.11.1.14 gcu_vector8_count()	133
5.11.1.15 gcu_vector8_create()	134
5.11.1.16 gcu_vector8_destroy()	134
5.12 test/test-debug.cpp File Reference	134
5.12.1 Detailed Description	135
5.13 test/test-hash.cpp File Reference	135
5.13.1 Detailed Description	137
5.14 test/test-type.cpp File Reference	137
5.14.1 Detailed Description	137
5.15 test/test-vector.cpp File Reference	138
5.15.1 Detailed Description	138
Index	139

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_Hash16_Cell	16-bit container holding the information for an entry in the hash table	7
GCU_Hash16_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_Hash16_Table	Container holding the information of the hash table	9
GCU_Hash16_Value	16-bit container used to return the result of looking for a hash in the hash table	10
GCU_Hash32_Cell	32-bit container holding the information for an entry in the hash table	11
GCU_Hash32_Iterator	A container used to hold the state of an iterator which can be used to traverse all elements of a hash table	12
GCU_Hash32_Table	32-bit container holding the information of the hash table	13
GCU_Hash32_Value	32-bit container used to return the result of looking for a hash in the hash table	14
GCU_Hash64_Cell	64-bit container holding the information for an entry in the hash table	15
GCU_Hash64_Iterator	A 64-bit container used to hold the state of an iterator which can be used to traverse all elements of a hash table	16
GCU_Hash64_Table	64-bit container holding the information of the hash table	17
GCU_Hash64_Value	64-bit container used to return the result of looking for a hash in the hash table	18
GCU_Hash8_Cell	8-bit container holding the information for an entry in the hash table	19
GCU_Hash8_Iterator	A container used to hold the state of an iterator which can be used to traverse all elements of a hash table	20
GCU_Hash8_Table	Container holding the information of the hash table	21
GCU_Hash8_Value	8-bit container used to return the result of looking for a hash in the hash table	22

GCU_Type16_Union	
A union of all basic, 16-bit types to be used by generic, 16-bit containers	23
GCU_Type32_Union	
A union of all basic, 32-bit types to be used by generic, 32-bit containers	23
GCU_Type64_Union	
A union of all basic, 64-bit types to be used by generic, 64-bit containers	24
GCU_Type8_Union	
A union of all basic, 8-bit types to be used by generic, 8-bit containers	24
GCU_Vector16	
Container holding the information of the 16-bit vector	25
GCU_Vector32	
Container holding the information of the 32-bit vector	26
GCU_Vector64	
Container holding the information of the 64-bit vector	27
GCU_Vector8	
Container holding the information of the 8-bit vector	28

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	29
include/cutil/ float.h	
Type definitions for float types	32
include/cutil/ hash.h	
A simple hash table implementation	33
include/cutil/ libver.h	
Header file used to control the version numbering and function namespace for all of the library	53
include/cutil/ type.h	
Type definitions and utilities for use by the Ghoti.io projects	56
include/cutil/ vector.h	
A simple vector implementation	74
src/ debug.c	84
src/ float_identifier.c	
Simple program to generate correct floating point type names for a given byte size	86
src/ hash.c	87
src/ type.c	106
src/ vector.c	126
test/ test-debug.cpp	
Test the behavior of Ghoti.io CUtil debug library and tools	134
test/ test-hash.cpp	
Test the behavior of Ghoti.io CUtil hash table library	135
test/ test-type.cpp	
Test the behavior of Ghoti.io CUtil type library	137
test/ test-vector.cpp	
Test the behavior of Ghoti.io CUtil hash table library	138

Chapter 4

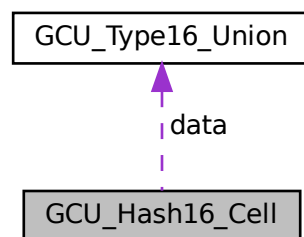
Class Documentation

4.1 GCU_Hash16_Cell Struct Reference

16-bit container holding the information for an entry in the hash table.

```
#include <hash.h>
```

Collaboration diagram for GCU_Hash16_Cell:



Public Attributes

- `size_t` `hash`
The hash of the entry.
- `GCU_Type16_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

16-bit 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:

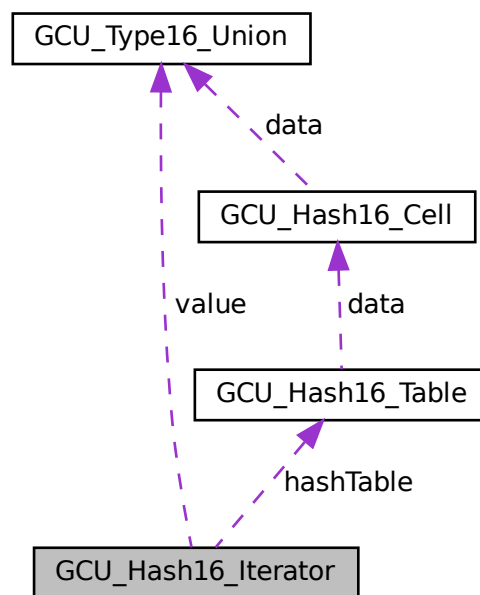
- `include/cutil/hash.h`

4.2 GCU_Hash16_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_Hash16_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_Type16_Union` `value`
The data pointed to by the iterator.
- `GCU_Hash16_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:

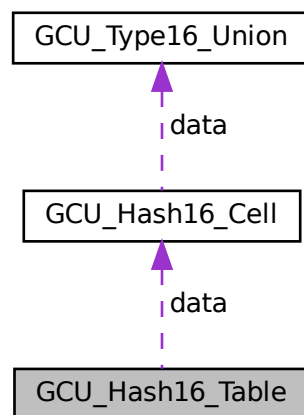
- `include/cutil/hash.h`

4.3 GCU_Hash16_Table Struct Reference

Container holding the information of the hash table.

```
#include <hash.h>
```

Collaboration diagram for GCU_Hash16_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_Hash16_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_hash16_create\(\)](#).
2. Destroy the has table using [gcu_hash16_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:

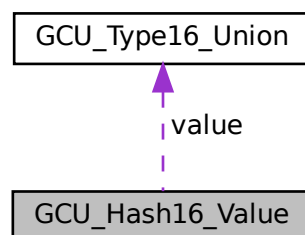
- [include/cutil/hash.h](#)

4.4 GCU_Hash16_Value Struct Reference

16-bit container used to return the result of looking for a hash in the hash table.

```
#include <hash.h>
```

Collaboration diagram for GCU_Hash16_Value:



Public Attributes

- `bool` [exists](#)
Whether or not the value exists in the hash table.
- [GCU_Type16_Union](#) `value`
The value found in the table (if it exists).

4.4.1 Detailed Description

16-bit 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 imperative 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:

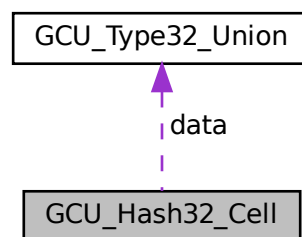
- `include/cutil/hash.h`

4.5 GCU_Hash32_Cell Struct Reference

32-bit container holding the information for an entry in the hash table.

```
#include <hash.h>
```

Collaboration diagram for GCU_Hash32_Cell:



Public Attributes

- `size_t hash`
The hash of the entry.
- `GCU_Type32_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.

4.5.1 Detailed Description

32-bit 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:

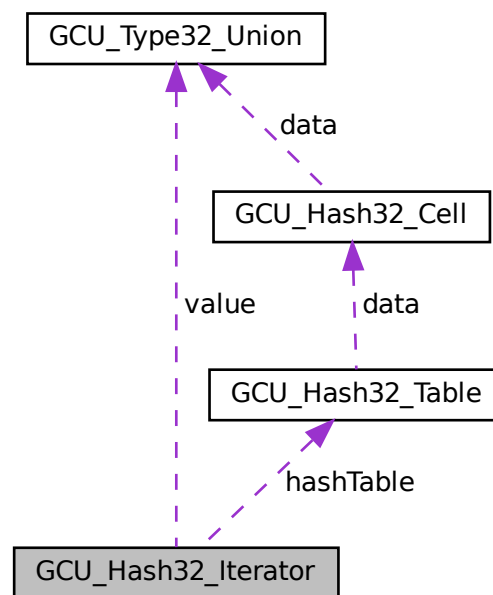
- `include/cutil/hash.h`

4.6 GCU_Hash32_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_Hash32_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_Type32_Union value`
The data pointed to by the iterator.
- `GCU_Hash32_Table * hashTable`
The hash table that the iterator traverses.

4.6.1 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:

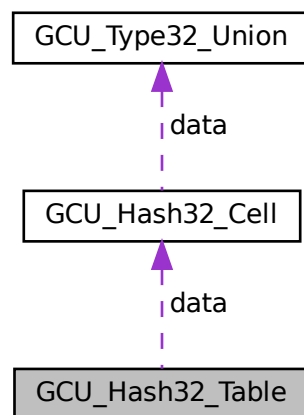
- `include/cutil/hash.h`

4.7 GCU_Hash32_Table Struct Reference

32-bit container holding the information of the hash table.

```
#include <hash.h>
```

Collaboration diagram for GCU_Hash32_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_Hash32_Cell * data`
A pointer to the array of data cells.

4.7.1 Detailed Description

32-bit 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_hash32_create\(\)](#).
2. Destroy the has table using [gcu_hash32_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:

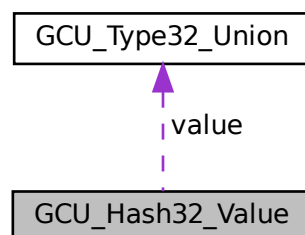
- [include/cutil/hash.h](#)

4.8 GCU_Hash32_Value Struct Reference

32-bit container used to return the result of looking for a hash in the hash table.

```
#include <hash.h>
```

Collaboration diagram for GCU_Hash32_Value:



Public Attributes

- `bool` [exists](#)
Whether or not the value exists in the hash table.
- [GCU_Type32_Union](#) [value](#)
The value found in the table (if it exists).

4.8.1 Detailed Description

32-bit 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 imperative 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:

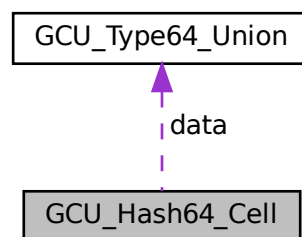
- `include/cutil/hash.h`

4.9 GCU_Hash64_Cell Struct Reference

64-bit container holding the information for an entry in the hash table.

```
#include <hash.h>
```

Collaboration diagram for GCU_Hash64_Cell:



Public Attributes

- `size_t hash`
The hash of the entry.
- `GCU_Type64_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.

4.9.1 Detailed Description

64-bit 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:

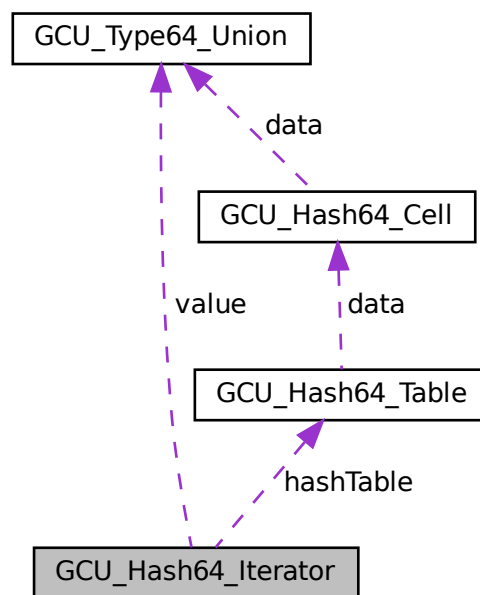
- `include/cutil/hash.h`

4.10 GCU_Hash64_Iterator Struct Reference

A 64-bit 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_Hash64_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_Type64_Union value`
The data pointed to by the iterator.
- `GCU_Hash64_Table * hashTable`
The hash table that the iterator traverses.

4.10.1 Detailed Description

A 64-bit 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:

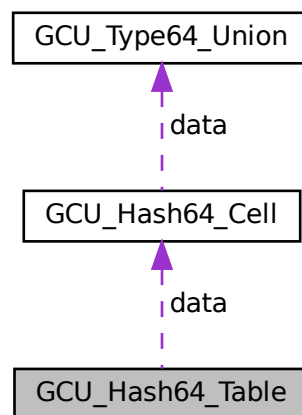
- `include/cutil/hash.h`

4.11 GCU_Hash64_Table Struct Reference

64-bit container holding the information of the hash table.

```
#include <hash.h>
```

Collaboration diagram for GCU_Hash64_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_Hash64_Cell * data`
A pointer to the array of data cells.

4.11.1 Detailed Description

64-bit 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:

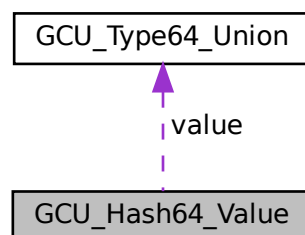
- `include/cutil/hash.h`

4.12 GCU_Hash64_Value Struct Reference

64-bit container used to return the result of looking for a hash in the hash table.

```
#include <hash.h>
```

Collaboration diagram for GCU_Hash64_Value:



Public Attributes

- `bool exists`
Whether or not the value exists in the hash table.
- `GCU_Type64_Union value`
The value found in the table (if it exists).

4.12.1 Detailed Description

64-bit 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 imperative 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:

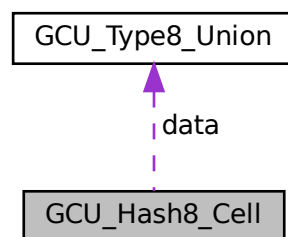
- `include/cutil/hash.h`

4.13 GCU_Hash8_Cell Struct Reference

8-bit container holding the information for an entry in the hash table.

```
#include <hash.h>
```

Collaboration diagram for GCU_Hash8_Cell:



Public Attributes

- `size_t hash`
The hash of the entry.
- `GCU_Type8_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.

4.13.1 Detailed Description

8-bit 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:

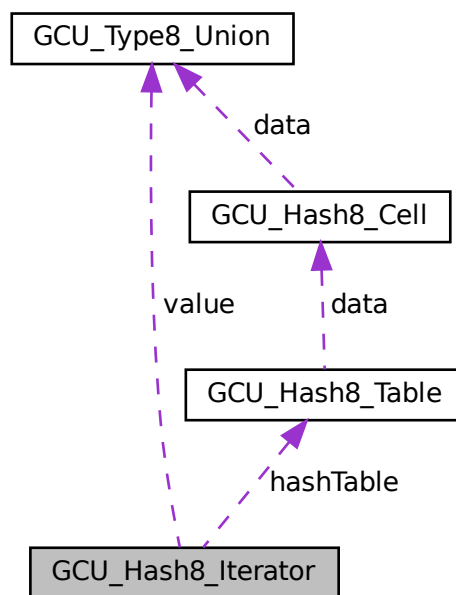
- `include/cutil/hash.h`

4.14 GCU_Hash8_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_Hash8_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_Type8_Union` `value`
The data pointed to by the iterator.
- `GCU_Hash8_Table *` `hashTable`
The hash table that the iterator traverses.

4.14.1 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:

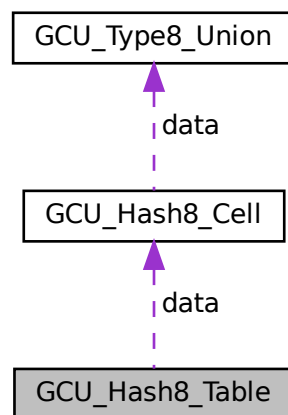
- `include/cutil/hash.h`

4.15 GCU_Hash8_Table Struct Reference

Container holding the information of the hash table.

```
#include <hash.h>
```

Collaboration diagram for GCU_Hash8_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_Hash8_Cell * data`
A pointer to the array of data cells.

4.15.1 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_hash8_create\(\)](#).
2. Destroy the has table using [gcu_hash8_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:

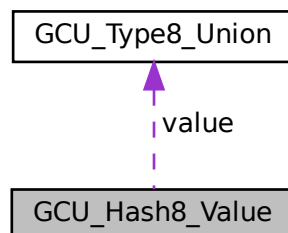
- [include/cutil/hash.h](#)

4.16 GCU_Hash8_Value Struct Reference

8-bit container used to return the result of looking for a hash in the hash table.

```
#include <hash.h>
```

Collaboration diagram for GCU_Hash8_Value:



Public Attributes

- [bool exists](#)
Whether or not the value exists in the hash table.
- [GCU_Type8_Union value](#)
The value found in the table (if it exists).

4.16.1 Detailed Description

8-bit 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 imperative 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`

4.17 GCU_Type16_Union Union Reference

A union of all basic, 16-bit types to be used by generic, 16-bit containers.

```
#include <type.h>
```

Public Attributes

- `uint16_t ui16`
- `uint8_t ui8`
- `int16_t i16`
- `int8_t i8`
- `char c`

4.17.1 Detailed Description

A union of all basic, 16-bit types to be used by generic, 16-bit containers.

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

- `include/cutil/type.h`

4.18 GCU_Type32_Union Union Reference

A union of all basic, 32-bit types to be used by generic, 32-bit containers.

```
#include <type.h>
```

Public Attributes

- `uint32_t ui32`
- `uint16_t ui16`
- `uint8_t ui8`
- `int32_t i32`
- `int16_t i16`
- `int8_t i8`
- `GCU_float32_t f32`
- `char c`

4.18.1 Detailed Description

A union of all basic, 32-bit types to be used by generic, 32-bit containers.

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

- [include/cutil/type.h](#)

4.19 GCU_Type64_Union Union Reference

A union of all basic, 64-bit types to be used by generic, 64-bit 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**
- GCU_float64_t **f64**
- GCU_float32_t **f32**
- char **c**

4.19.1 Detailed Description

A union of all basic, 64-bit types to be used by generic, 64-bit containers.

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

- [include/cutil/type.h](#)

4.20 GCU_Type8_Union Union Reference

A union of all basic, 8-bit types to be used by generic, 8-bit containers.

```
#include <type.h>
```

Public Attributes

- uint8_t **ui8**
- int8_t **i8**
- char **c**

4.20.1 Detailed Description

A union of all basic, 8-bit types to be used by generic, 8-bit containers.

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

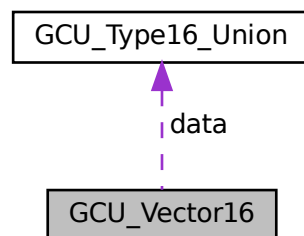
- [include/cutil/type.h](#)

4.21 GCU_Vector16 Struct Reference

Container holding the information of the 16-bit vector.

```
#include <vector.h>
```

Collaboration diagram for GCU_Vector16:



Public Attributes

- `size_t` [capacity](#)
The total item capacity of the vector.
- `size_t` [count](#)
The count of non-empty cells.
- [GCU_Type16_Union](#) * [data](#)
A pointer to the array of data cells.

4.21.1 Detailed Description

Container holding the information of the 16-bit vector.

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

1. Initialize the vector using [gcu_vector16_create\(\)](#).
2. Destroy the vector using [gcu_vector16_destroy\(\)](#).
3. Implementation of any thread-safety synchronization.
4. Life cycle management of the contents of the vector. The vector 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:

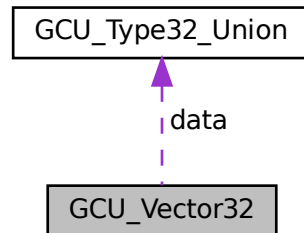
- [include/cutil/vector.h](#)

4.22 GCU_Vector32 Struct Reference

Container holding the information of the 32-bit vector.

```
#include <vector.h>
```

Collaboration diagram for GCU_Vector32:



Public Attributes

- `size_t` [capacity](#)
The total item capacity of the vector.
- `size_t` [count](#)
The count of non-empty cells.
- `GCU_Type32_Union *` [data](#)
A pointer to the array of data cells.

4.22.1 Detailed Description

Container holding the information of the 32-bit vector.

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

1. Initialize the vector using [gcu_vector32_create\(\)](#).
2. Destroy the vector using [gcu_vector32_destroy\(\)](#).
3. Implementation of any thread-safety synchronization.
4. Life cycle management of the contents of the vector. The vector 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:

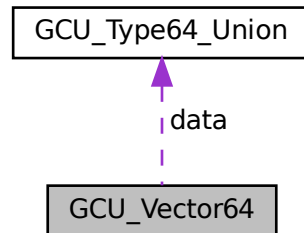
- `include/cutil/`[vector.h](#)

4.23 GCU_Vector64 Struct Reference

Container holding the information of the 64-bit vector.

```
#include <vector.h>
```

Collaboration diagram for GCU_Vector64:



Public Attributes

- `size_t capacity`
The total item capacity of the vector.
- `size_t count`
The count of non-empty cells.
- `GCU_Type64_Union * data`
A pointer to the array of data cells.

4.23.1 Detailed Description

Container holding the information of the 64-bit vector.

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

1. Initialize the vector using `gcu_vector64_create()`.
2. Destroy the vector using `gcu_vector64_destroy()`.
3. Implementation of any thread-safety synchronization.
4. Life cycle management of the contents of the vector. The vector 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:

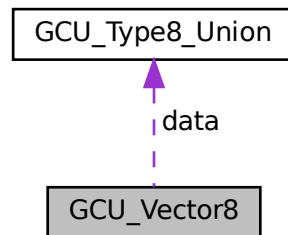
- `include/cutil/vector.h`

4.24 GCU_Vector8 Struct Reference

Container holding the information of the 8-bit vector.

```
#include <vector.h>
```

Collaboration diagram for GCU_Vector8:



Public Attributes

- `size_t` [capacity](#)
The total item capacity of the vector.
- `size_t` [count](#)
The count of non-empty cells.
- `GCU_Type8_Union *` [data](#)
A pointer to the array of data cells.

4.24.1 Detailed Description

Container holding the information of the 8-bit vector.

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

1. Initialize the vector using [gcu_vector8_create\(\)](#).
2. Destroy the vector using [gcu_vector8_destroy\(\)](#).
3. Implementation of any thread-safety synchronization.
4. Life cycle management of the contents of the vector. The vector 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/`[vector.h](#)

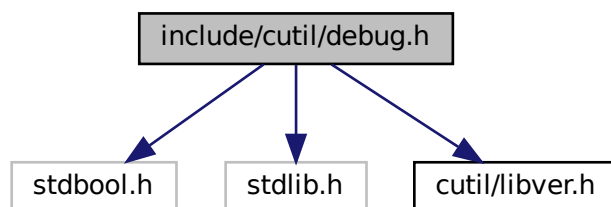
Chapter 5

File Documentation

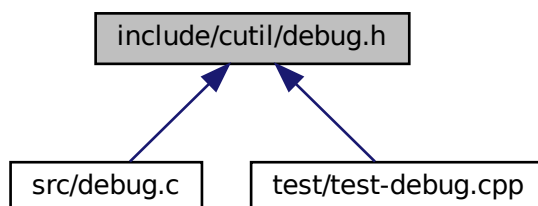
5.1 include/cutil/debug.h File Reference

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

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

<i>pointer</i>	The beginning byte of the currently allocated memory.
<i>file</i>	The name of the file from which the function was called.
<i>line</i>	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

<i>size</i>	The number of bytes requested.
<i>file</i>	The name of the file from which the function was called.
<i>line</i>	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

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

Returns

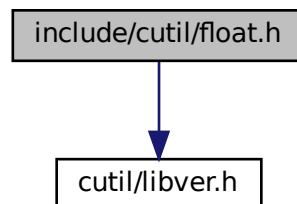
The beginning byte of the reallocated memory.

5.2 include/cutil/float.h File Reference

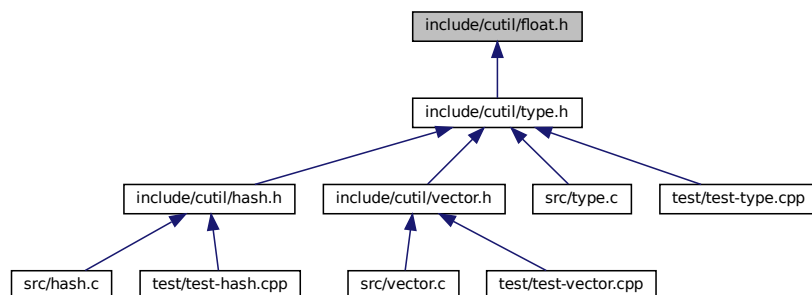
Type definitions for float types.

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

Include dependency graph for float.h:



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



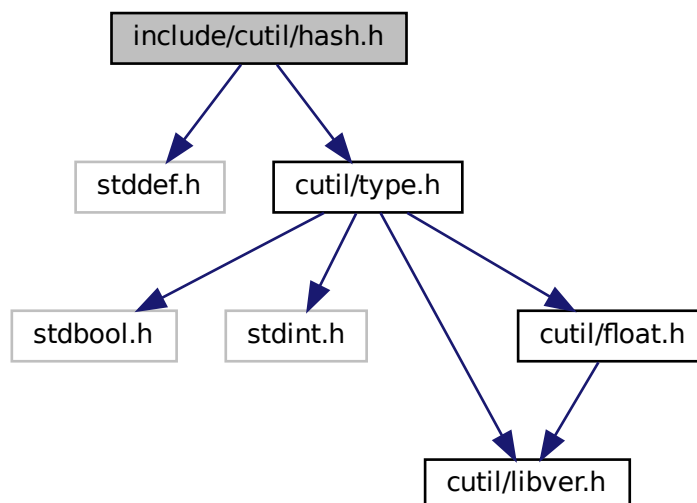
5.2.1 Detailed Description

Type definitions for float types.

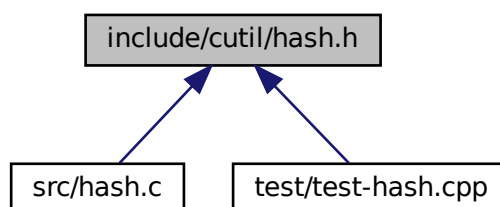
5.3 include/cutil/hash.h File Reference

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_Hash64_Value](#)
64-bit container used to return the result of looking for a hash in the hash table.
- struct [GCU_Hash64_Cell](#)
64-bit container holding the information for an entry in the hash table.
- struct [GCU_Hash64_Table](#)
64-bit container holding the information of the hash table.
- struct [GCU_Hash64_Iterator](#)
A 64-bit container used to hold the state of an iterator which can be used to traverse all elements of a hash table.
- struct [GCU_Hash32_Value](#)
32-bit container used to return the result of looking for a hash in the hash table.
- struct [GCU_Hash32_Cell](#)
32-bit container holding the information for an entry in the hash table.
- struct [GCU_Hash32_Table](#)
32-bit container holding the information of the hash table.
- struct [GCU_Hash32_Iterator](#)
A container used to hold the state of an iterator which can be used to traverse all elements of a hash table.
- struct [GCU_Hash16_Value](#)
16-bit container used to return the result of looking for a hash in the hash table.
- struct [GCU_Hash16_Cell](#)
16-bit container holding the information for an entry in the hash table.
- struct [GCU_Hash16_Table](#)
Container holding the information of the hash table.
- struct [GCU_Hash16_Iterator](#)
A container used to hold the state of an iterator which can be used to traverse all elements of a hash table.
- struct [GCU_Hash8_Value](#)
8-bit container used to return the result of looking for a hash in the hash table.
- struct [GCU_Hash8_Cell](#)
8-bit container holding the information for an entry in the hash table.
- struct [GCU_Hash8_Table](#)
Container holding the information of the hash table.
- struct [GCU_Hash8_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_Hash64_Table](#) * [gcu_hash64_create](#) (size_t count)
Create a hash table structure for 64-bit entries.
- void [gcu_hash64_destroy](#) ([GCU_Hash64_Table](#) *hashTable)
Destroy a hash table structure and clean up memory allocations.
- bool [gcu_hash64_set](#) ([GCU_Hash64_Table](#) *hashTable, size_t hash, [GCU_Type64_Union](#) value)
Set a value in the hash table.
- [GCU_Hash64_Value](#) [gcu_hash64_get](#) ([GCU_Hash64_Table](#) *hashTable, size_t hash)
Get a value from the hash table (if it exists).
- bool [gcu_hash64_contains](#) ([GCU_Hash64_Table](#) *hashTable, size_t hash)
Check to see whether or not a hash table contains a specific hash.
- bool [gcu_hash64_remove](#) ([GCU_Hash64_Table](#) *hashTable, size_t hash)
Remove a hash from the table.
- size_t [gcu_hash64_count](#) ([GCU_Hash64_Table](#) *hashTable)

- Get a count of active entries in the hash table.*

 - [GCU_Hash64_Iterator](#) [gcu_hash64_iterator_get](#) ([GCU_Hash64_Table](#) *hashTable)

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

 - [GCU_Hash64_Iterator](#) [gcu_hash64_iterator_next](#) ([GCU_Hash64_Iterator](#) iterator)

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

 - [GCU_Hash32_Table](#) * [gcu_hash32_create](#) (size_t count)

Create a hash table structure for 32-bit entries.

 - void [gcu_hash32_destroy](#) ([GCU_Hash32_Table](#) *hashTable)

Destroy a hash table structure and clean up memory allocations.

 - bool [gcu_hash32_set](#) ([GCU_Hash32_Table](#) *hashTable, size_t hash, [GCU_Type32_Union](#) value)

Set a value in the hash table.

 - [GCU_Hash32_Value](#) [gcu_hash32_get](#) ([GCU_Hash32_Table](#) *hashTable, size_t hash)

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

 - bool [gcu_hash32_contains](#) ([GCU_Hash32_Table](#) *hashTable, size_t hash)

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

 - bool [gcu_hash32_remove](#) ([GCU_Hash32_Table](#) *hashTable, size_t hash)

Remove a hash from the table.

 - size_t [gcu_hash32_count](#) ([GCU_Hash32_Table](#) *hashTable)

Get a count of active entries in the hash table.

 - [GCU_Hash32_Iterator](#) [gcu_hash32_iterator_get](#) ([GCU_Hash32_Table](#) *hashTable)

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

 - [GCU_Hash32_Iterator](#) [gcu_hash32_iterator_next](#) ([GCU_Hash32_Iterator](#) iterator)

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

 - [GCU_Hash16_Table](#) * [gcu_hash16_create](#) (size_t count)

Create a hash table structure.

 - void [gcu_hash16_destroy](#) ([GCU_Hash16_Table](#) *hashTable)

Destroy a hash table structure and clean up memory allocations.

 - bool [gcu_hash16_set](#) ([GCU_Hash16_Table](#) *hashTable, size_t hash, [GCU_Type16_Union](#) value)

Set a value in the hash table.

 - [GCU_Hash16_Value](#) [gcu_hash16_get](#) ([GCU_Hash16_Table](#) *hashTable, size_t hash)

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

 - bool [gcu_hash16_contains](#) ([GCU_Hash16_Table](#) *hashTable, size_t hash)

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

 - bool [gcu_hash16_remove](#) ([GCU_Hash16_Table](#) *hashTable, size_t hash)

Remove a hash from the table.

 - size_t [gcu_hash16_count](#) ([GCU_Hash16_Table](#) *hashTable)

Get a count of active entries in the hash table.

 - [GCU_Hash16_Iterator](#) [gcu_hash16_iterator_get](#) ([GCU_Hash16_Table](#) *hashTable)

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

 - [GCU_Hash16_Iterator](#) [gcu_hash16_iterator_next](#) ([GCU_Hash16_Iterator](#) iterator)

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

 - [GCU_Hash8_Table](#) * [gcu_hash8_create](#) (size_t count)

Create a hash table structure.

 - void [gcu_hash8_destroy](#) ([GCU_Hash8_Table](#) *hashTable)

Destroy a hash table structure and clean up memory allocations.

 - bool [gcu_hash8_set](#) ([GCU_Hash8_Table](#) *hashTable, size_t hash, [GCU_Type8_Union](#) value)

Set a value in the hash table.

 - [GCU_Hash8_Value](#) [gcu_hash8_get](#) ([GCU_Hash8_Table](#) *hashTable, size_t hash)

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

 - bool [gcu_hash8_contains](#) ([GCU_Hash8_Table](#) *hashTable, size_t hash)

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

- `bool gcu_hash8_remove (GCU_Hash8_Table *hashTable, size_t hash)`
Remove a hash from the table.
- `size_t gcu_hash8_count (GCU_Hash8_Table *hashTable)`
Get a count of active entries in the hash table.
- `GCU_Hash8_Iterator gcu_hash8_iterator_get (GCU_Hash8_Table *hashTable)`
Get an iterator which can be used to iterate through the entries of the hash table.
- `GCU_Hash8_Iterator gcu_hash8_iterator_next (GCU_Hash8_Iterator iterator)`
Get an iterator to the next element in the table (if it exists).

5.3.1 Detailed Description

A simple hash table implementation.

5.3.2 Function Documentation

5.3.2.1 gcu_hash16_contains()

```
bool gcu_hash16_contains (
    GCU_Hash16_Table * hashTable,
    size_t hash )
```

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

Parameters

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

Returns

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

5.3.2.2 gcu_hash16_count()

```
size_t gcu_hash16_count (
    GCU_Hash16_Table * hashTable )
```

Get a count of active entries in the hash table.

Parameters

<i>hashTable</i>	The hash table structure on which to operate.
------------------	---

Returns

The count of active entries in the hash table.

5.3.2.3 gcu_hash16_create()

```
GCU_Hash16_Table* gcu_hash16_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

<i>count</i>	The number of items anticipated to be stored in the hash table.
--------------	---

Returns

A struct containing the hash table information.

5.3.2.4 gcu_hash16_destroy()

```
void gcu_hash16_destroy (
    GCU_Hash16_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

<i>hashTable</i>	The hash table structure to be destroyed.
------------------	---

5.3.2.5 gcu_hash16_get()

```
GCU_Hash16_Value gcu_hash16_get (
```

```
GCU_Hash16_Table * hashTable,
size_t hash )
```

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

Parameters

<i>hashTable</i>	The hash table structure on which to operate.
<i>hash</i>	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).

5.3.2.6 gcu_hash16_iterator_get()

```
GCU_Hash16_Iterator gcu_hash16_iterator_get (
    GCU_Hash16_Table * hashTable )
```

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

Parameters

<i>hashTable</i>	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:



5.3.2.7 gcu_hash16_iterator_next()

```
GCU_Hash16_Iterator gcu_hash16_iterator_next (
    GCU_Hash16_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_hash16_set\(\)](#), therefore, should be considered as an invalidation of any iterators associated with the hash table.

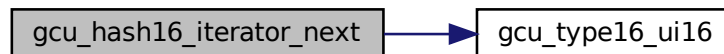
Parameters

<i>iterator</i>	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:



5.3.2.8 gcu_hash16_remove()

```
bool gcu_hash16_remove (
    GCU_Hash16_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

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

Returns

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

5.3.2.9 gcu_hash16_set()

```
bool gcu_hash16_set (
    GCU_Hash16_Table * hashTable,
    size_t hash,
    GCU_Type16_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_hash16_create()`.

Parameters

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

Returns

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

5.3.2.10 gcu_hash32_contains()

```
bool gcu_hash32_contains (
    GCU_Hash32_Table * hashTable,
    size_t hash )
```

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

Parameters

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

Returns

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

5.3.2.11 gcu_hash32_count()

```
size_t gcu_hash32_count (
    GCU_Hash32_Table * hashTable )
```

Get a count of active entries in the hash table.

Parameters

<i>hashTable</i>	The hash table structure on which to operate.
------------------	---

Returns

The count of active entries in the hash table.

5.3.2.12 gcu_hash32_create()

```
GCU_Hash32_Table* gcu_hash32_create (
    size_t count )
```

Create a hash table structure for 32-bit entries.

All invocations of a hash table must have a corresponding [gcu_hash32_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

<i>count</i>	The number of items anticipated to be stored in the hash table.
--------------	---

Returns

A struct containing the hash table information.

5.3.2.13 gcu_hash32_destroy()

```
void gcu_hash32_destroy (
    GCU_Hash32_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

<i>hashTable</i>	The hash table structure to be destroyed.
------------------	---

5.3.2.14 gcu_hash32_get()

```
GCU_Hash32_Value gcu_hash32_get (
    GCU_Hash32_Table * hashTable,
    size_t hash )
```

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

Parameters

<i>hashTable</i>	The hash table structure on which to operate.
<i>hash</i>	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).

5.3.2.15 gcu_hash32_iterator_get()

```
GCU_Hash32_Iterator gcu_hash32_iterator_get (
    GCU_Hash32_Table * hashTable )
```

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

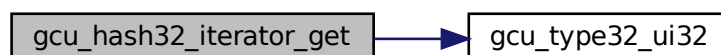
Parameters

<i>hashTable</i>	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:



5.3.2.16 gcu_hash32_iterator_next()

```
GCU_Hash32_Iterator gcu_hash32_iterator_next (
    GCU_Hash32_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_hash32_set\(\)](#), therefore, should be considered as an invalidation of any iterators associated with the hash table.

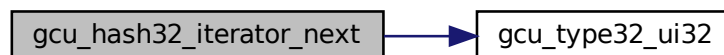
Parameters

<i>iterator</i>	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:



5.3.2.17 gcu_hash32_remove()

```
bool gcu_hash32_remove (
    GCU_Hash32_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

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

Returns

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

5.3.2.18 gcu_hash32_set()

```
bool gcu_hash32_set (
    GCU_Hash32_Table * hashTable,
    size_t hash,
    GCU_Type32_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_hash32_create\(\)](#).

Parameters

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

Returns

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

5.3.2.19 gcu_hash64_contains()

```
bool gcu_hash64_contains (
    GCU_Hash64_Table * hashTable,
    size_t hash )
```

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

Parameters

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

Returns

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

5.3.2.20 `gcu_hash64_count()`

```
size_t gcu_hash64_count (
    GCU_Hash64_Table * hashTable )
```

Get a count of active entries in the hash table.

Parameters

<code>hashTable</code>	The hash table structure on which to operate.
------------------------	---

Returns

The count of active entries in the hash table.

5.3.2.21 `gcu_hash64_create()`

```
GCU_Hash64_Table* gcu_hash64_create (
    size_t count )
```

Create a hash table structure for 64-bit entries.

All invocations of a hash table must have a corresponding `gcu_hash64_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

<code>count</code>	The number of items anticipated to be stored in the hash table.
--------------------	---

Returns

A struct containing the hash table information.

5.3.2.22 `gcu_hash64_destroy()`

```
void gcu_hash64_destroy (
    GCU_Hash64_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

<i>hashTable</i>	The hash table structure to be destroyed.
------------------	---

5.3.2.23 gcu_hash64_get()

```
GCU_Hash64_Value gcu_hash64_get (
    GCU_Hash64_Table * hashTable,
    size_t hash )
```

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

Parameters

<i>hashTable</i>	The hash table structure on which to operate.
<i>hash</i>	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).

5.3.2.24 gcu_hash64_iterator_get()

```
GCU_Hash64_Iterator gcu_hash64_iterator_get (
    GCU_Hash64_Table * hashTable )
```

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

Parameters

<i>hashTable</i>	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:

**5.3.2.25 gcu_hash64_iterator_next()**

```
GCU_Hash64_Iterator gcu_hash64_iterator_next (
    GCU_Hash64_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_hash64_set\(\)](#), therefore, should be considered as an invalidation of any iterators associated with the hash table.

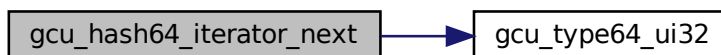
Parameters

<i>iterator</i>	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:



5.3.2.26 `gcu_hash64_remove()`

```
bool gcu_hash64_remove (
    GCU_Hash64_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

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

Returns

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

5.3.2.27 `gcu_hash64_set()`

```
bool gcu_hash64_set (
    GCU_Hash64_Table * hashTable,
    size_t hash,
    GCU_Type64_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

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

Returns

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

5.3.2.28 `gcu_hash8_contains()`

```
bool gcu_hash8_contains (
    GCU_Hash8_Table * hashTable,
    size_t hash )
```

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

Parameters

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

Returns

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

5.3.2.29 gcu_hash8_count()

```
size_t gcu_hash8_count (
    GCU_Hash8_Table * hashTable )
```

Get a count of active entries in the hash table.

Parameters

<i>hashTable</i>	The hash table structure on which to operate.
------------------	---

Returns

The count of active entries in the hash table.

5.3.2.30 gcu_hash8_create()

```
GCU_Hash8_Table* gcu_hash8_create (
    size_t count )
```

Create a hash table structure.

All invocations of a hash table must have a corresponding [gcu_hash8_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

<i>count</i>	The number of items anticipated to be stored in the hash table.
--------------	---

Returns

A struct containing the hash table information.

5.3.2.31 gcu_hash8_destroy()

```
void gcu_hash8_destroy (
    GCU_Hash8_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

<i>hashTable</i>	The hash table structure to be destroyed.
------------------	---

5.3.2.32 gcu_hash8_get()

```
GCU_Hash8_Value gcu_hash8_get (
    GCU_Hash8_Table * hashTable,
    size_t hash )
```

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

Parameters

<i>hashTable</i>	The hash table structure on which to operate.
<i>hash</i>	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).

5.3.2.33 gcu_hash8_iterator_get()

```
GCU_Hash8_Iterator gcu_hash8_iterator_get (
    GCU_Hash8_Table * hashTable )
```

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

Parameters

<i>hashTable</i>	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:

**5.3.2.34 gcu_hash8_iterator_next()**

```
GCU_Hash8_Iterator gcu_hash8_iterator_next (  
    GCU_Hash8_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_hash8_set\(\)](#), therefore, should be considered as an invalidation of any iterators associated with the hash table.

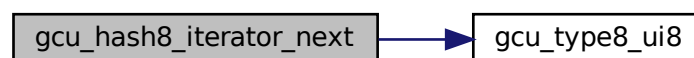
Parameters

<i>iterator</i>	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:



5.3.2.35 gcu_hash8_remove()

```
bool gcu_hash8_remove (
    GCU_Hash8_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

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

Returns

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

5.3.2.36 gcu_hash8_set()

```
bool gcu_hash8_set (
    GCU_Hash8_Table * hashTable,
    size_t hash,
    GCU_Type8_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_hash8_create\(\)](#).

Parameters

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

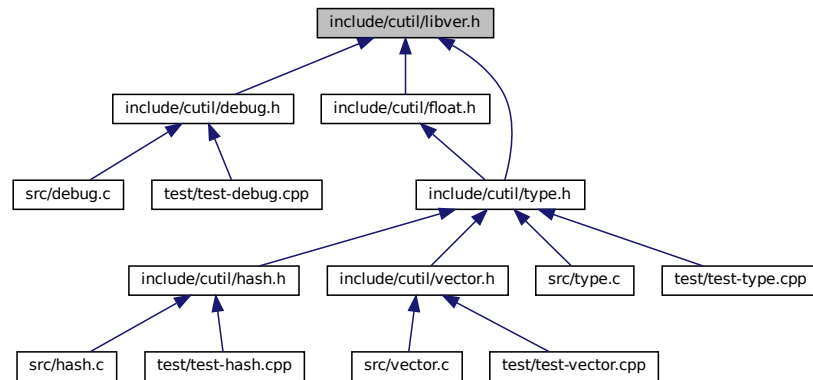
Returns

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

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

5.4.1 Detailed Description

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

5.4.2 Macro Definition Documentation

5.4.2.1 GHOTIIO_CUTIL

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

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

Parameters

<i>NAME</i>	The name which will be prepended with the GHOTIIO_CUTIL_NAME.
-------------	---

5.4.2.2 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

<i>a</i>	The first part of the identifier.
<i>b</i>	The second part of the identifier.

Returns

A call to the GHOTIIO_CUTIL_CONCAT_INNER() macro.

5.4.2.3 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

<i>a</i>	The first part of the identifier.
<i>b</i>	The second part of the identifier.

Returns

The concatenation of a to b.

5.4.2.4 GHOTIIO_CUTIL_NAME

```
#define GHOTIIO_CUTIL_NAME ghotiio_cutil_dev
```

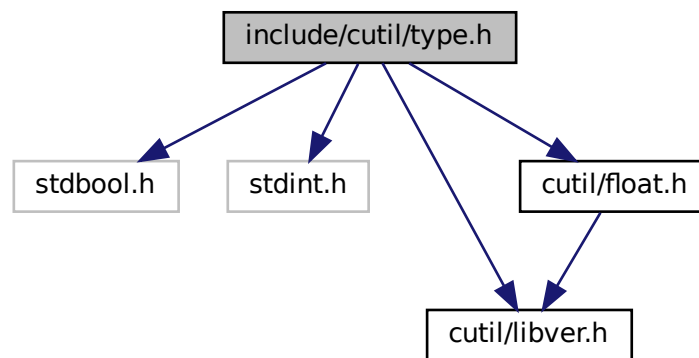
Used in conjunction with the GHOTIIO_CUTIL...

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

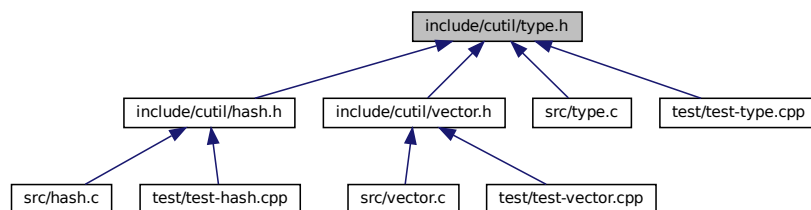
5.5 include/cutil/type.h File Reference

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

```
#include <stdbool.h>
#include <stdint.h>
#include "cutil/libver.h"
#include "cutil/float.h"
Include dependency graph for type.h:
```



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



Classes

- union [GCU_Type64_Union](#)
A union of all basic, 64-bit types to be used by generic, 64-bit containers.
- union [GCU_Type32_Union](#)
A union of all basic, 32-bit types to be used by generic, 32-bit containers.
- union [GCU_Type16_Union](#)
A union of all basic, 16-bit types to be used by generic, 16-bit containers.
- union [GCU_Type8_Union](#)
A union of all basic, 8-bit types to be used by generic, 8-bit containers.

Functions

- [GCU_Type64_Union gcu_type64_p](#) (void *val)
Create a 64-bit union variable with the type `void *`.
- [GCU_Type64_Union gcu_type64_ui64](#) (uint64_t val)
Create a 64-bit union variable with the type `uint64_t`.
- [GCU_Type64_Union gcu_type64_ui32](#) (uint32_t val)
Create a 64-bit union variable with the type `uint32_t`.
- [GCU_Type64_Union gcu_type64_ui16](#) (uint16_t val)
Create a 64-bit union variable with the type `uint16_t`.
- [GCU_Type64_Union gcu_type64_ui8](#) (uint8_t val)
Create a 64-bit union variable with the type `uint8_t`.
- [GCU_Type64_Union gcu_type64_i64](#) (int64_t val)
Create a 64-bit union variable with the type `int64_t`.
- [GCU_Type64_Union gcu_type64_i32](#) (int32_t val)
Create a 64-bit union variable with the type `int32_t`.
- [GCU_Type64_Union gcu_type64_i16](#) (int16_t val)
Create a 64-bit union variable with the type `int16_t`.
- [GCU_Type64_Union gcu_type64_i8](#) (int8_t val)
Create a 64-bit union variable with the type `int8_t`.
- [GCU_Type64_Union gcu_type64_f64](#) (GCU_float64_t val)
Create a 64-bit union variable with the type `float` with 64 bits.
- [GCU_Type64_Union gcu_type64_f32](#) (GCU_float32_t val)
Create a 64-bit union variable with the type `float` with 32 bits.
- [GCU_Type64_Union gcu_type64_c](#) (char val)
Create a 64-bit union variable with the type `char`.
- [GCU_Type32_Union gcu_type32_ui64](#) (uint64_t val)
Create a 32-bit union variable with the type `uint64_t`.
- [GCU_Type32_Union gcu_type32_ui32](#) (uint32_t val)
Create a 32-bit union variable with the type `uint32_t`.
- [GCU_Type32_Union gcu_type32_ui16](#) (uint16_t val)
Create a 32-bit union variable with the type `uint16_t`.
- [GCU_Type32_Union gcu_type32_ui8](#) (uint8_t val)
Create a 32-bit union variable with the type `uint8_t`.
- [GCU_Type32_Union gcu_type32_i32](#) (int32_t val)
Create a 32-bit union variable with the type `int32_t`.
- [GCU_Type32_Union gcu_type32_i16](#) (int16_t val)
Create a 32-bit union variable with the type `int16_t`.
- [GCU_Type32_Union gcu_type32_i8](#) (int8_t val)

- Create a 32-bit union variable with the type `int8_t`.*
 - `GCU_Type32_Union gcu_type32_f32` (`GCU_float32_t` val)
- Create a 32-bit union variable with the type `float` with 32 bits.*
 - `GCU_Type32_Union gcu_type32_c` (`char` val)
- Create a 32-bit union variable with the type `char`.*
 - `GCU_Type16_Union gcu_type16_ui16` (`uint16_t` val)
- Create a 16-bit union variable with the type `uint16_t`.*
 - `GCU_Type16_Union gcu_type16_ui8` (`uint8_t` val)
- Create a 16-bit union variable with the type `uint8_t`.*
 - `GCU_Type16_Union gcu_type16_i16` (`int16_t` val)
- Create a 16-bit union variable with the type `int16_t`.*
 - `GCU_Type16_Union gcu_type16_i8` (`int8_t` val)
- Create a 16-bit union variable with the type `int8_t`.*
 - `GCU_Type16_Union gcu_type16_c` (`char` val)
- Create a 16-bit union variable with the type `char`.*
 - `GCU_Type8_Union gcu_type8_ui8` (`uint8_t` val)
- Create a 8-bit union variable with the type `uint8_t`.*
 - `GCU_Type8_Union gcu_type8_i8` (`int8_t` val)
- Create a 8-bit union variable with the type `int8_t`.*
 - `GCU_Type8_Union gcu_type8_c` (`char` val)
- Create a 8-bit union variable with the type `char`.*

5.5.1 Detailed Description

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

5.5.2 Function Documentation

5.5.2.1 `gcu_type16_c()`

```
GCU_Type16_Union gcu_type16_c (
    char val )
```

Create a 16-bit union variable with the type `char`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.5.2.2 gcu_type16_i16()

```
GCU_Type16_Union gcu_type16_i16 (  
    int16_t val )
```

Create a 16-bit union variable with the type `int16_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.5.2.3 gcu_type16_i8()

```
GCU_Type16_Union gcu_type16_i8 (  
    int8_t val )
```

Create a 16-bit union variable with the type `int8_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.5.2.4 gcu_type16_ui16()

```
GCU_Type16_Union gcu_type16_ui16 (  
    uint16_t val )
```

Create a 16-bit union variable with the type `uint16_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.5.2.5 gcu_type16_ui8()

```
GCU_Type16_Union gcu_type16_ui8 (  
    uint8_t val )
```

Create a 16-bit union variable with the type `uint8_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.5.2.6 gcu_type32_c()

```
GCU_Type32_Union gcu_type32_c (  
    char val )
```

Create a 32-bit union variable with the type `char`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.5.2.7 gcu_type32_f32()

```
GCU_Type32_Union gcu_type32_f32 (  
    GCU_float32_t val )
```

Create a 32-bit union variable with the type float with 32 bits.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.5.2.8 gcu_type32_i16()

```
GCU_Type32_Union gcu_type32_i16 (  
    int16_t val )
```

Create a 32-bit union variable with the type `int16_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.5.2.9 gcu_type32_i32()

```
GCU_Type32_Union gcu_type32_i32 (  
    int32_t val )
```

Create a 32-bit union variable with the type `int32_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.5.2.10 gcu_type32_i8()

```
GCU_Type32_Union gcu_type32_i8 (  
    int8_t val )
```

Create a 32-bit union variable with the type `int8_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.5.2.11 gcu_type32_ui16()

```
GCU_Type32_Union gcu_type32_ui16 (  
    uint16_t val )
```

Create a 32-bit union variable with the type `uint16_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.5.2.12 gcu_type32_ui32()

```
GCU_Type32_Union gcu_type32_ui32 (  
    uint32_t val )
```

Create a 32-bit union variable with the type `uint32_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.5.2.13 gcu_type32_ui64()

```
GCU_Type32_Union gcu_type32_ui64 (  
    uint64_t val )
```


Create a 32-bit union variable with the type `uint64_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.5.2.14 gcu_type32_ui8()

```
GCU_Type32_Union gcu_type32_ui8 (  
    uint8_t val )
```

Create a 32-bit union variable with the type `uint8_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.5.2.15 gcu_type64_c()

```
GCU_Type64_Union gcu_type64_c (  
    char val )
```

Create a 64-bit union variable with the type `char`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.5.2.16 gcu_type64_f32()

```
GCU_Type64_Union gcu_type64_f32 (  
    GCU_float32_t val )
```

Create a 64-bit union variable with the type float with 32 bits.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.5.2.17 gcu_type64_f64()

```
GCU_Type64_Union gcu_type64_f64 (
    GCU_float64_t val )
```

Create a 64-bit union variable with the type float with 64 bits.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.5.2.18 gcu_type64_i16()

```
GCU_Type64_Union gcu_type64_i16 (
    int16_t val )
```

Create a 64-bit union variable with the type int16_t.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.5.2.19 gcu_type64_i32()

```
GCU_Type64_Union gcu_type64_i32 (
    int32_t val )
```

Create a 64-bit union variable with the type `int32_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.5.2.20 gcu_type64_i64()

```
GCU_Type64_Union gcu_type64_i64 (  
    int64_t val )
```

Create a 64-bit union variable with the type `int64_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.5.2.21 gcu_type64_i8()

```
GCU_Type64_Union gcu_type64_i8 (  
    int8_t val )
```

Create a 64-bit union variable with the type `int8_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.5.2.22 gcu_type64_p()

```
GCU_Type64_Union gcu_type64_p (  
    void * val )
```

Create a 64-bit union variable with the type `void *`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.5.2.23 gcu_type64_ui16()

```
GCU_Type64_Union gcu_type64_ui16 (  
    uint16_t val )
```

Create a 64-bit union variable with the type `uint16_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.5.2.24 gcu_type64_ui32()

```
GCU_Type64_Union gcu_type64_ui32 (  
    uint32_t val )
```

Create a 64-bit union variable with the type `uint32_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.5.2.25 gcu_type64_ui64()

```
GCU_Type64_Union gcu_type64_ui64 (  
    uint64_t val )
```


Create a 64-bit union variable with the type `uint64_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.5.2.26 gcu_type64_ui8()

```
GCU_Type64_Union gcu_type64_ui8 (  
    uint8_t val )
```

Create a 64-bit union variable with the type `uint8_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.5.2.27 gcu_type8_c()

```
GCU_Type8_Union gcu_type8_c (  
    char val )
```

Create a 8-bit union variable with the type `char`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.5.2.28 gcu_type8_i8()

```
GCU_Type8_Union gcu_type8_i8 (  
    int8_t val )
```

Create a 8-bit union variable with the type `int8_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.5.2.29 gcu_type8_ui8()

```
GCU_Type8_Union gcu_type8_ui8 (  
    uint8_t val )
```

Create a 8-bit union variable with the type `uint8_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

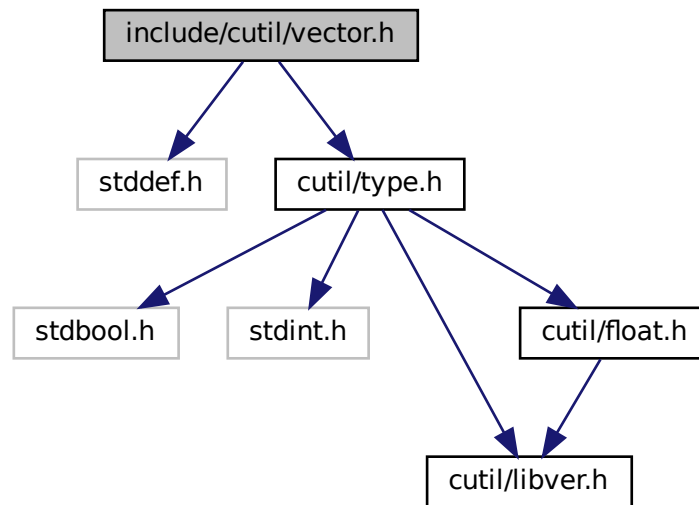
The union variable.

5.6 include/cutil/vector.h File Reference

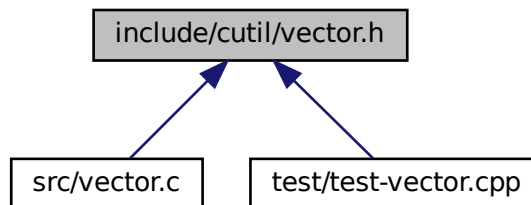
A simple vector implementation.

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

Include dependency graph for vector.h:



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



Classes

- struct [GCU_Vector64](#)
Container holding the information of the 64-bit vector.
- struct [GCU_Vector32](#)
Container holding the information of the 32-bit vector.
- struct [GCU_Vector16](#)
Container holding the information of the 16-bit vector.
- struct [GCU_Vector8](#)
Container holding the information of the 8-bit vector.

Functions

- `GCU_Vector64 * gcu_vector64_create (size_t count)`
Create a vector structure.
- `void gcu_vector64_destroy (GCU_Vector64 *vector)`
Destroy a vector structure and clean up memory allocations.
- `bool gcu_vector64_append (GCU_Vector64 *vector, GCU_Type64_Union value)`
Append an item at the end of the vector.
- `size_t gcu_vector64_count (GCU_Vector64 *vector)`
Get a count of entries in the vector.
- `GCU_Vector32 * gcu_vector32_create (size_t count)`
Create a vector structure.
- `void gcu_vector32_destroy (GCU_Vector32 *vector)`
Destroy a vector structure and clean up memory allocations.
- `bool gcu_vector32_append (GCU_Vector32 *vector, GCU_Type32_Union value)`
Append an item at the end of the vector.
- `size_t gcu_vector32_count (GCU_Vector32 *vector)`
Get a count of entries in the vector.
- `GCU_Vector16 * gcu_vector16_create (size_t count)`
Create a vector structure.
- `void gcu_vector16_destroy (GCU_Vector16 *vector)`
Destroy a vector structure and clean up memory allocations.
- `bool gcu_vector16_append (GCU_Vector16 *vector, GCU_Type16_Union value)`
Append an item at the end of the vector.
- `size_t gcu_vector16_count (GCU_Vector16 *vector)`
Get a count of entries in the vector.
- `GCU_Vector8 * gcu_vector8_create (size_t count)`
Create a vector structure.
- `void gcu_vector8_destroy (GCU_Vector8 *vector)`
Destroy a vector structure and clean up memory allocations.
- `bool gcu_vector8_append (GCU_Vector8 *vector, GCU_Type8_Union value)`
Append an item at the end of the vector.
- `size_t gcu_vector8_count (GCU_Vector8 *vector)`
Get a count of entries in the vector.

5.6.1 Detailed Description

A simple vector implementation.

5.6.2 Function Documentation

5.6.2.1 `gcu_vector16_append()`

```
bool gcu_vector16_append (
    GCU_Vector16 * vector,
    GCU_Type16_Union value )
```

Append an item at the end of the vector.

If there is not enough space in the current data structure, new space will be attempted to be allocated. This may invalidate any pointers to the previous data locations.

Parameters

<i>value</i>	The item to append to the end of the vector.
--------------	--

Returns

`true` on success, `false` otherwise.

5.6.2.2 gcu_vector16_count()

```
size_t gcu_vector16_count (
    GCU_Vector16 * vector )
```

Get a count of entries in the vector.

Parameters

<i>vector</i>	The vector structure on which to operate.
---------------	---

Returns

The count of entries in the vector.

5.6.2.3 gcu_vector16_create()

```
GCU_Vector16* gcu_vector16_create (
    size_t count )
```

Create a vector structure.

All invocations of a vector must have a corresponding `gcu_vector16_destroy()` call in order to clean up dynamically-allocated memory.

The vector 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 vector.

Parameters

<i>count</i>	The number of items anticipated to be stored in the vector.
--------------	---

Returns

A struct containing the vector information.

5.6.2.4 gcu_vector16_destroy()

```
void gcu_vector16_destroy (
    GCU_Vector16 * vector )
```

Destroy a vector 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

<i>vector</i>	The vector structure to be destroyed.
---------------	---------------------------------------

5.6.2.5 gcu_vector32_append()

```
bool gcu_vector32_append (
    GCU_Vector32 * vector,
    GCU_Type32_Union value )
```

Append an item at the end of the vector.

If there is not enough space in the current data structure, new space will be attempted to be allocated. This may invalidate any pointers to the previous data locations.

Parameters

<i>value</i>	The item to append to the end of the vector.
--------------	--

Returns

`true` on success, `false` otherwise.

5.6.2.6 gcu_vector32_count()

```
size_t gcu_vector32_count (
    GCU_Vector32 * vector )
```

Get a count of entries in the vector.

Parameters

<code>vector</code>	The vector structure on which to operate.
---------------------	---

Returns

The count of entries in the vector.

5.6.2.7 `gcu_vector32_create()`

```
GCU_Vector32* gcu_vector32_create (
    size_t count )
```

Create a vector structure.

All invocations of a vector must have a corresponding `gcu_vector32_destroy()` call in order to clean up dynamically-allocated memory.

The vector 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 vector.

Parameters

<code>count</code>	The number of items anticipated to be stored in the vector.
--------------------	---

Returns

A struct containing the vector information.

5.6.2.8 `gcu_vector32_destroy()`

```
void gcu_vector32_destroy (
    GCU_Vector32 * vector )
```

Destroy a vector 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

<code>vector</code>	The vector structure to be destroyed.
---------------------	---------------------------------------

5.6.2.9 gcu_vector64_append()

```
bool gcu_vector64_append (
    GCU_Vector64 * vector,
    GCU_Type64_Union value )
```

Append an item at the end of the vector.

If there is not enough space in the current data structure, new space will be attempted to be allocated. This may invalidate any pointers to the previous data locations.

Parameters

<i>value</i>	The item to append to the end of the vector.
--------------	--

Returns

`true` on success, `false` otherwise.

5.6.2.10 gcu_vector64_count()

```
size_t gcu_vector64_count (
    GCU_Vector64 * vector )
```

Get a count of entries in the vector.

Parameters

<i>vector</i>	The vector structure on which to operate.
---------------	---

Returns

The count of entries in the vector.

5.6.2.11 gcu_vector64_create()

```
GCU_Vector64* gcu_vector64_create (
    size_t count )
```

Create a vector structure.

All invocations of a vector must have a corresponding [gcu_vector64_destroy\(\)](#) call in order to clean up dynamically-allocated memory.

The vector 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 vector.

Parameters

<i>count</i>	The number of items anticipated to be stored in the vector.
--------------	---

Returns

A struct containing the vector information.

5.6.2.12 gcu_vector64_destroy()

```
void gcu_vector64_destroy (
    GCU_Vector64 * vector )
```

Destroy a vector 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

<i>vector</i>	The vector structure to be destroyed.
---------------	---------------------------------------

5.6.2.13 gcu_vector8_append()

```
bool gcu_vector8_append (
    GCU_Vector8 * vector,
    GCU_Type8_Union value )
```

Append an item at the end of the vector.

If there is not enough space in the current data structure, new space will be attempted to be allocated. This may invalidate any pointers to the previous data locations.

Parameters

<i>value</i>	The item to append to the end of the vector.
--------------	--

Returns

`true` on success, `false` otherwise.

5.6.2.14 `gcu_vector8_count()`

```
size_t gcu_vector8_count (
    GCU_Vector8 * vector )
```

Get a count of entries in the vector.

Parameters

<code>vector</code>	The vector structure on which to operate.
---------------------	---

Returns

The count of entries in the vector.

5.6.2.15 `gcu_vector8_create()`

```
GCU_Vector8* gcu_vector8_create (
    size_t count )
```

Create a vector structure.

All invocations of a vector must have a corresponding `gcu_vector8_destroy()` call in order to clean up dynamically-allocated memory.

The vector 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 vector.

Parameters

<code>count</code>	The number of items anticipated to be stored in the vector.
--------------------	---

Returns

A struct containing the vector information.

5.6.2.16 `gcu_vector8_destroy()`

```
void gcu_vector8_destroy (
    GCU_Vector8 * vector )
```

Destroy a vector structure and clean up memory allocations.

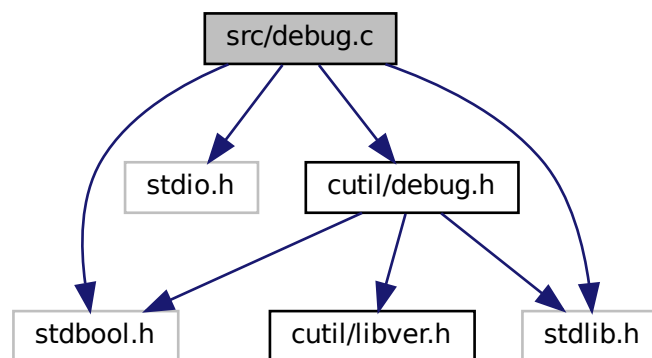
This function will not address any memory allocations of the elements themselves (if any). The programmer is responsible for controlling any memory management on behalf of the elements.

Parameters

<code>vector</code>	The vector structure to be destroyed.
---------------------	---------------------------------------

5.7 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

5.7.1 Function Documentation

5.7.1.1 gcu_calloc()

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

Wrapper for the standard `calloc()` function.

Parameters

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

Returns

The beginning byte of the allocated memory.

5.7.1.2 gcu_free()

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

Wrapper for the standard `free()` function.

Parameters

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

5.7.1.3 gcu_malloc()

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

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

Wrapper for the standard malloc() function.

Parameters

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

Returns

The beginning byte of the allocated memory.

5.7.1.4 gcu_realloc()

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

Wrapper for the standard realloc() function.

Parameters

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

Returns

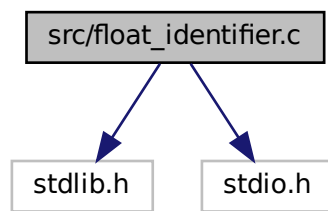
The beginning byte of the reallocated memory.

5.8 src/float_identifier.c File Reference

Simple program to generate correct floating point type names for a given byte size.

```
#include <stdlib.h>
#include <stdio.h>
```


Include dependency graph for float_identifier.c:



Functions

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

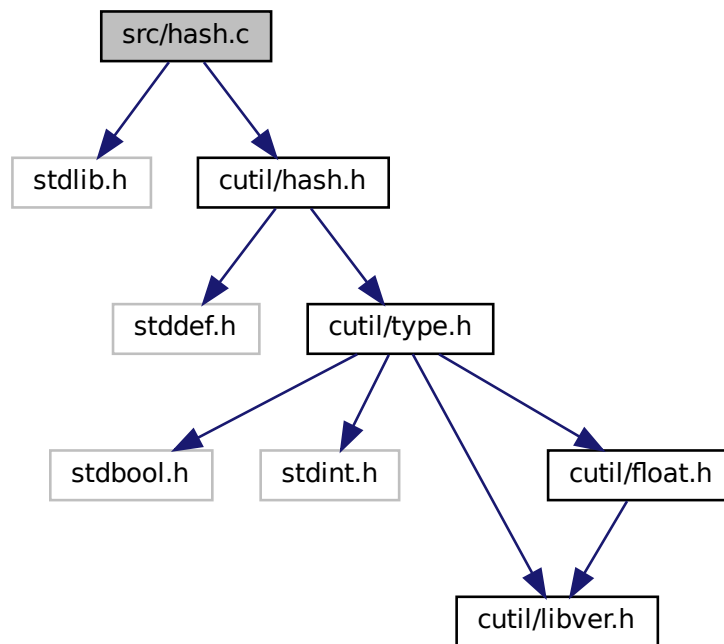
5.8.1 Detailed Description

Simple program to generate correct floating point type names for a given byte size.

5.9 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_Hash64_Table * gcu_hash64_create (size_t count)`
Create a hash table structure for 64-bit entries.
- `void gcu_hash64_destroy (GCU_Hash64_Table *hashTable)`
Destroy a hash table structure and clean up memory allocations.
- `bool gcu_hash64_set (GCU_Hash64_Table *hashTable, size_t hash, GCU_Type64_Union value)`
Set a value in the hash table.
- `GCU_Hash64_Value gcu_hash64_get (GCU_Hash64_Table *hashTable, size_t hash)`
Get a value from the hash table (if it exists).
- `bool gcu_hash64_contains (GCU_Hash64_Table *hashTable, size_t hash)`
Check to see whether or not a hash table contains a specific hash.
- `bool gcu_hash64_remove (GCU_Hash64_Table *hashTable, size_t hash)`
Remove a hash from the table.
- `size_t gcu_hash64_count (GCU_Hash64_Table *hashTable)`
Get a count of active entries in the hash table.
- `GCU_Hash64_Iterator gcu_hash64_iterator_get (GCU_Hash64_Table *hashTable)`
Get an iterator which can be used to iterate through the entries of the hash table.
- `GCU_Hash64_Iterator gcu_hash64_iterator_next (GCU_Hash64_Iterator iterator)`

- Get an iterator to the next element in the table (if it exists).*
- [GCU_Hash32_Table](#) * [gcu_hash32_create](#) (size_t count)
Create a hash table structure for 32-bit entries.
- void [gcu_hash32_destroy](#) ([GCU_Hash32_Table](#) *hashTable)
Destroy a hash table structure and clean up memory allocations.
- bool [gcu_hash32_set](#) ([GCU_Hash32_Table](#) *hashTable, size_t hash, [GCU_Type32_Union](#) value)
Set a value in the hash table.
- [GCU_Hash32_Value](#) [gcu_hash32_get](#) ([GCU_Hash32_Table](#) *hashTable, size_t hash)
Get a value from the hash table (if it exists).
- bool [gcu_hash32_contains](#) ([GCU_Hash32_Table](#) *hashTable, size_t hash)
Check to see whether or not a hash table contains a specific hash.
- bool [gcu_hash32_remove](#) ([GCU_Hash32_Table](#) *hashTable, size_t hash)
Remove a hash from the table.
- size_t [gcu_hash32_count](#) ([GCU_Hash32_Table](#) *hashTable)
Get a count of active entries in the hash table.
- [GCU_Hash32_Iterator](#) [gcu_hash32_iterator_get](#) ([GCU_Hash32_Table](#) *hashTable)
Get an iterator which can be used to iterate through the entries of the hash table.
- [GCU_Hash32_Iterator](#) [gcu_hash32_iterator_next](#) ([GCU_Hash32_Iterator](#) iterator)
Get an iterator to the next element in the table (if it exists).
- [GCU_Hash16_Table](#) * [gcu_hash16_create](#) (size_t count)
Create a hash table structure.
- void [gcu_hash16_destroy](#) ([GCU_Hash16_Table](#) *hashTable)
Destroy a hash table structure and clean up memory allocations.
- bool [gcu_hash16_set](#) ([GCU_Hash16_Table](#) *hashTable, size_t hash, [GCU_Type16_Union](#) value)
Set a value in the hash table.
- [GCU_Hash16_Value](#) [gcu_hash16_get](#) ([GCU_Hash16_Table](#) *hashTable, size_t hash)
Get a value from the hash table (if it exists).
- bool [gcu_hash16_contains](#) ([GCU_Hash16_Table](#) *hashTable, size_t hash)
Check to see whether or not a hash table contains a specific hash.
- bool [gcu_hash16_remove](#) ([GCU_Hash16_Table](#) *hashTable, size_t hash)
Remove a hash from the table.
- size_t [gcu_hash16_count](#) ([GCU_Hash16_Table](#) *hashTable)
Get a count of active entries in the hash table.
- [GCU_Hash16_Iterator](#) [gcu_hash16_iterator_get](#) ([GCU_Hash16_Table](#) *hashTable)
Get an iterator which can be used to iterate through the entries of the hash table.
- [GCU_Hash16_Iterator](#) [gcu_hash16_iterator_next](#) ([GCU_Hash16_Iterator](#) iterator)
Get an iterator to the next element in the table (if it exists).
- [GCU_Hash8_Table](#) * [gcu_hash8_create](#) (size_t count)
Create a hash table structure.
- void [gcu_hash8_destroy](#) ([GCU_Hash8_Table](#) *hashTable)
Destroy a hash table structure and clean up memory allocations.
- bool [gcu_hash8_set](#) ([GCU_Hash8_Table](#) *hashTable, size_t hash, [GCU_Type8_Union](#) value)
Set a value in the hash table.
- [GCU_Hash8_Value](#) [gcu_hash8_get](#) ([GCU_Hash8_Table](#) *hashTable, size_t hash)
Get a value from the hash table (if it exists).
- bool [gcu_hash8_contains](#) ([GCU_Hash8_Table](#) *hashTable, size_t hash)
Check to see whether or not a hash table contains a specific hash.
- bool [gcu_hash8_remove](#) ([GCU_Hash8_Table](#) *hashTable, size_t hash)
Remove a hash from the table.
- size_t [gcu_hash8_count](#) ([GCU_Hash8_Table](#) *hashTable)
Get a count of active entries in the hash table.

- [GCU_Hash8_Iterator](#) [gcu_hash8_iterator_get](#) ([GCU_Hash8_Table](#) *hashTable)
Get an iterator which can be used to iterate through the entries of the hash table.
- [GCU_Hash8_Iterator](#) [gcu_hash8_iterator_next](#) ([GCU_Hash8_Iterator](#) iterator)
Get an iterator to the next element in the table (if it exists).

5.9.1 Function Documentation

5.9.1.1 [gcu_hash16_contains\(\)](#)

```
bool gcu_hash16_contains (
    GCU\_Hash16\_Table * hashTable,
    size_t hash )
```

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

Parameters

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

Returns

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

5.9.1.2 [gcu_hash16_count\(\)](#)

```
size_t gcu_hash16_count (
    GCU\_Hash16\_Table * hashTable )
```

Get a count of active entries in the hash table.

Parameters

<i>hashTable</i>	The hash table structure on which to operate.
------------------	---

Returns

The count of active entries in the hash table.

5.9.1.3 gcu_hash16_create()

```
GCU_Hash16_Table* gcu_hash16_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 accommodate 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

<i>count</i>	The number of items anticipated to be stored in the hash table.
--------------	---

Returns

A struct containing the hash table information.

5.9.1.4 gcu_hash16_destroy()

```
void gcu_hash16_destroy (
    GCU_Hash16_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

<i>hashTable</i>	The hash table structure to be destroyed.
------------------	---

5.9.1.5 gcu_hash16_get()

```
GCU_Hash16_Value gcu_hash16_get (
    GCU_Hash16_Table * hashTable,
    size_t hash )
```

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

Parameters

<i>hashTable</i>	The hash table structure on which to operate.
<i>hash</i>	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).

5.9.1.6 gcu_hash16_iterator_get()

```
GCU_Hash16_Iterator gcu_hash16_iterator_get (
    GCU_Hash16_Table * hashTable )
```

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

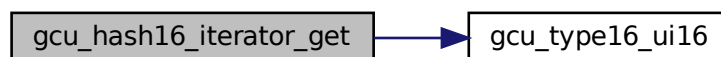
Parameters

<i>hashTable</i>	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:

**5.9.1.7 gcu_hash16_iterator_next()**

```
GCU_Hash16_Iterator gcu_hash16_iterator_next (
    GCU_Hash16_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_hash16_set\(\)](#), therefore, should be considered as an invalidation of any iterators associated with the hash table.

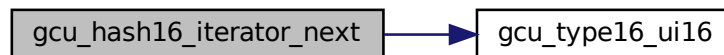
Parameters

<i>iterator</i>	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:

**5.9.1.8 gcu_hash16_remove()**

```
bool gcu_hash16_remove (  
    GCU_Hash16_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

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

Returns

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

5.9.1.9 gcu_hash16_set()

```
bool gcu_hash16_set (  
    GCU_Hash16_Table * hashTable,
```

```
size_t hash,  
GCU_Type16_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_hash16_create()`.

Parameters

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

Returns

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

5.9.1.10 gcu_hash32_contains()

```
bool gcu_hash32_contains (   
    GCU_Hash32_Table * hashTable,  
    size_t hash )
```

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

Parameters

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

Returns

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

5.9.1.11 gcu_hash32_count()

```
size_t gcu_hash32_count (   
    GCU_Hash32_Table * hashTable )
```

Get a count of active entries in the hash table.

Parameters

<i>hashTable</i>	The hash table structure on which to operate.
------------------	---

Returns

The count of active entries in the hash table.

5.9.1.12 gcu_hash32_create()

```
GCU_Hash32_Table* gcu_hash32_create (
    size_t count )
```

Create a hash table structure for 32-bit entries.

All invocations of a hash table must have a corresponding [gcu_hash32_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

<i>count</i>	The number of items anticipated to be stored in the hash table.
--------------	---

Returns

A struct containing the hash table information.

5.9.1.13 gcu_hash32_destroy()

```
void gcu_hash32_destroy (
    GCU_Hash32_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

<i>hashTable</i>	The hash table structure to be destroyed.
------------------	---

5.9.1.14 gcu_hash32_get()

```
GCU_Hash32_Value gcu_hash32_get (
```

```
GCU_Hash32_Table * hashTable,
size_t hash )
```

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

Parameters

<i>hashTable</i>	The hash table structure on which to operate.
<i>hash</i>	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).

5.9.1.15 gcu_hash32_iterator_get()

```
GCU_Hash32_Iterator gcu_hash32_iterator_get (
    GCU_Hash32_Table * hashTable )
```

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

Parameters

<i>hashTable</i>	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:



5.9.1.16 gcu_hash32_iterator_next()

```
GCU_Hash32_Iterator gcu_hash32_iterator_next (
    GCU_Hash32_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_hash32_set\(\)](#), therefore, should be considered as an invalidation of any iterators associated with the hash table.

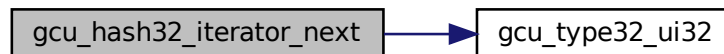
Parameters

<i>iterator</i>	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:



5.9.1.17 gcu_hash32_remove()

```
bool gcu_hash32_remove (
    GCU_Hash32_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

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

Returns

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

5.9.1.18 `gcu_hash32_set()`

```
bool gcu_hash32_set (
    GCU_Hash32_Table * hashTable,
    size_t hash,
    GCU_Type32_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_hash32_create()`.

Parameters

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

Returns

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

5.9.1.19 `gcu_hash64_contains()`

```
bool gcu_hash64_contains (
    GCU_Hash64_Table * hashTable,
    size_t hash )
```

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

Parameters

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

Returns

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

5.9.1.20 `gcu_hash64_count()`

```
size_t gcu_hash64_count (
    GCU_Hash64_Table * hashTable )
```

Get a count of active entries in the hash table.

Parameters

<i>hashTable</i>	The hash table structure on which to operate.
------------------	---

Returns

The count of active entries in the hash table.

5.9.1.21 gcu_hash64_create()

```
GCU_Hash64_Table* gcu_hash64_create (
    size_t count )
```

Create a hash table structure for 64-bit entries.

All invocations of a hash table must have a corresponding [gcu_hash64_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

<i>count</i>	The number of items anticipated to be stored in the hash table.
--------------	---

Returns

A struct containing the hash table information.

5.9.1.22 gcu_hash64_destroy()

```
void gcu_hash64_destroy (
    GCU_Hash64_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

<i>hashTable</i>	The hash table structure to be destroyed.
------------------	---

5.9.1.23 gcu_hash64_get()

```
GCU_Hash64_Value gcu_hash64_get (
    GCU_Hash64_Table * hashTable,
    size_t hash )
```

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

Parameters

<i>hashTable</i>	The hash table structure on which to operate.
<i>hash</i>	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).

5.9.1.24 gcu_hash64_iterator_get()

```
GCU_Hash64_Iterator gcu_hash64_iterator_get (
    GCU_Hash64_Table * hashTable )
```

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

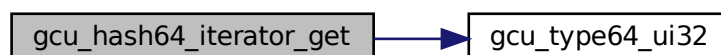
Parameters

<i>hashTable</i>	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:



5.9.1.25 gcu_hash64_iterator_next()

```
GCU_Hash64_Iterator gcu_hash64_iterator_next (
    GCU_Hash64_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_hash64_set\(\)](#), therefore, should be considered as an invalidation of any iterators associated with the hash table.

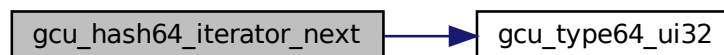
Parameters

<i>iterator</i>	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:



5.9.1.26 gcu_hash64_remove()

```
bool gcu_hash64_remove (
    GCU_Hash64_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

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

Returns

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

5.9.1.27 gcu_hash64_set()

```
bool gcu_hash64_set (
    GCU_Hash64_Table * hashTable,
    size_t hash,
    GCU_Type64_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

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

Returns

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

5.9.1.28 gcu_hash8_contains()

```
bool gcu_hash8_contains (
    GCU_Hash8_Table * hashTable,
    size_t hash )
```

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

Parameters

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

Returns

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

5.9.1.29 gcu_hash8_count()

```
size_t gcu_hash8_count (
    GCU_Hash8_Table * hashTable )
```

Get a count of active entries in the hash table.

Parameters

<i>hashTable</i>	The hash table structure on which to operate.
------------------	---

Returns

The count of active entries in the hash table.

5.9.1.30 gcu_hash8_create()

```
GCU_Hash8_Table* gcu_hash8_create (
    size_t count )
```

Create a hash table structure.

All invocations of a hash table must have a corresponding [gcu_hash8_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

<i>count</i>	The number of items anticipated to be stored in the hash table.
--------------	---

Returns

A struct containing the hash table information.

5.9.1.31 gcu_hash8_destroy()

```
void gcu_hash8_destroy (
    GCU_Hash8_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

<i>hashTable</i>	The hash table structure to be destroyed.
------------------	---

5.9.1.32 gcu_hash8_get()

```
GCU_Hash8_Value gcu_hash8_get (
    GCU_Hash8_Table * hashTable,
    size_t hash )
```

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

Parameters

<i>hashTable</i>	The hash table structure on which to operate.
<i>hash</i>	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).

5.9.1.33 gcu_hash8_iterator_get()

```
GCU_Hash8_Iterator gcu_hash8_iterator_get (
    GCU_Hash8_Table * hashTable )
```

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

Parameters

<i>hashTable</i>	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:

**5.9.1.34 gcu_hash8_iterator_next()**

```
GCU_Hash8_Iterator gcu_hash8_iterator_next (  
    GCU_Hash8_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_hash8_set\(\)](#), therefore, should be considered as an invalidation of any iterators associated with the hash table.

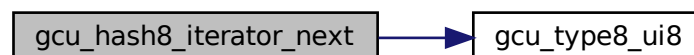
Parameters

<i>iterator</i>	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:



5.9.1.35 `gcu_hash8_remove()`

```
bool gcu_hash8_remove (
    GCU_Hash8_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

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

Returns

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

5.9.1.36 `gcu_hash8_set()`

```
bool gcu_hash8_set (
    GCU_Hash8_Table * hashTable,
    size_t hash,
    GCU_Type8_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_hash8_create()`.

Parameters

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

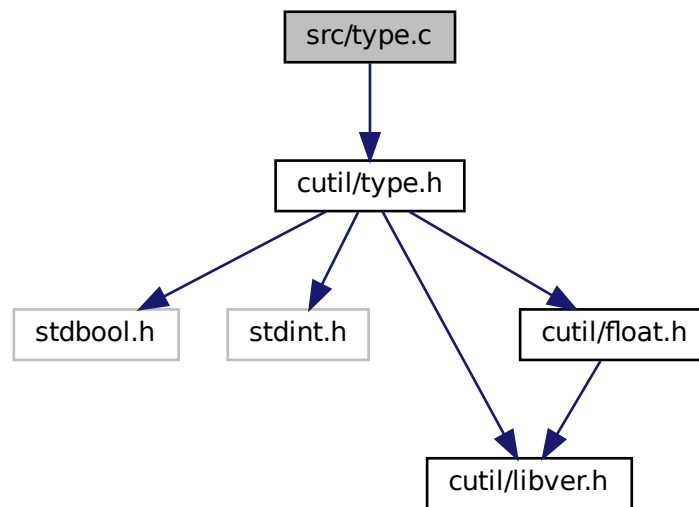
Returns

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

5.10 `src/type.c` File Reference

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

Include dependency graph for type.c:



Functions

- [GCU_Type64_Union gcu_type64_p](#) (void *val)
Create a 64-bit union variable with the type `void *`.
- [GCU_Type64_Union gcu_type64_ui64](#) (uint64_t val)
Create a 64-bit union variable with the type `uint64_t`.
- [GCU_Type64_Union gcu_type64_ui32](#) (uint32_t val)
Create a 64-bit union variable with the type `uint32_t`.
- [GCU_Type64_Union gcu_type64_ui16](#) (uint16_t val)
Create a 64-bit union variable with the type `uint16_t`.
- [GCU_Type64_Union gcu_type64_ui8](#) (uint8_t val)
Create a 64-bit union variable with the type `uint8_t`.
- [GCU_Type64_Union gcu_type64_i64](#) (int64_t val)
Create a 64-bit union variable with the type `int64_t`.
- [GCU_Type64_Union gcu_type64_i32](#) (int32_t val)
Create a 64-bit union variable with the type `int32_t`.
- [GCU_Type64_Union gcu_type64_i16](#) (int16_t val)
Create a 64-bit union variable with the type `int16_t`.
- [GCU_Type64_Union gcu_type64_i8](#) (int8_t val)
Create a 64-bit union variable with the type `int8_t`.
- [GCU_Type64_Union gcu_type64_f64](#) (GCU_float64_t val)
Create a 64-bit union variable with the type `float` with 64 bits.
- [GCU_Type64_Union gcu_type64_f32](#) (GCU_float32_t val)
Create a 64-bit union variable with the type `float` with 32 bits.
- [GCU_Type64_Union gcu_type64_c](#) (char val)
Create a 64-bit union variable with the type `char`.
- [GCU_Type32_Union gcu_type32_ui32](#) (uint32_t val)

- Create a 32-bit union variable with the type `uint32_t`.*
 - [GCU_Type32_Union gcu_type32_ui16](#) (`uint16_t val`)
- Create a 32-bit union variable with the type `uint16_t`.*
 - [GCU_Type32_Union gcu_type32_ui8](#) (`uint8_t val`)
- Create a 32-bit union variable with the type `uint8_t`.*
 - [GCU_Type32_Union gcu_type32_i32](#) (`int32_t val`)
- Create a 32-bit union variable with the type `int32_t`.*
 - [GCU_Type32_Union gcu_type32_i16](#) (`int16_t val`)
- Create a 32-bit union variable with the type `int16_t`.*
 - [GCU_Type32_Union gcu_type32_i8](#) (`int8_t val`)
- Create a 32-bit union variable with the type `int8_t`.*
 - [GCU_Type32_Union gcu_type32_f32](#) (`GCU_float32_t val`)
- Create a 32-bit union variable with the type `float` with 32 bits.*
 - [GCU_Type32_Union gcu_type32_c](#) (`char val`)
- Create a 32-bit union variable with the type `char`.*
 - [GCU_Type16_Union gcu_type16_ui16](#) (`uint16_t val`)
- Create a 16-bit union variable with the type `uint16_t`.*
 - [GCU_Type16_Union gcu_type16_ui8](#) (`uint8_t val`)
- Create a 16-bit union variable with the type `uint8_t`.*
 - [GCU_Type16_Union gcu_type16_i16](#) (`int16_t val`)
- Create a 16-bit union variable with the type `int16_t`.*
 - [GCU_Type16_Union gcu_type16_i8](#) (`int8_t val`)
- Create a 16-bit union variable with the type `int8_t`.*
 - [GCU_Type16_Union gcu_type16_c](#) (`char val`)
- Create a 16-bit union variable with the type `char`.*
 - [GCU_Type8_Union gcu_type8_ui8](#) (`uint8_t val`)
- Create a 8-bit union variable with the type `uint8_t`.*
 - [GCU_Type8_Union gcu_type8_i8](#) (`int8_t val`)
- Create a 8-bit union variable with the type `int8_t`.*
 - [GCU_Type8_Union gcu_type8_c](#) (`char val`)
- Create a 8-bit union variable with the type `char`.*

5.10.1 Function Documentation

5.10.1.1 `gcu_type16_c()`

```
GCU_Type16_Union gcu_type16_c (
    char val )
```

Create a 16-bit union variable with the type `char`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.10.1.2 gcu_type16_i16()

```
GCU_Type16_Union gcu_type16_i16 (  
    int16_t val )
```

Create a 16-bit union variable with the type `int16_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.10.1.3 gcu_type16_i8()

```
GCU_Type16_Union gcu_type16_i8 (  
    int8_t val )
```

Create a 16-bit union variable with the type `int8_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.10.1.4 gcu_type16_ui16()

```
GCU_Type16_Union gcu_type16_ui16 (  
    uint16_t val )
```

Create a 16-bit union variable with the type `uint16_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.10.1.5 gcu_type16_ui8()

```
GCU_Type16_Union gcu_type16_ui8 (  
    uint8_t val )
```

Create a 16-bit union variable with the type `uint8_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.10.1.6 gcu_type32_c()

```
GCU_Type32_Union gcu_type32_c (  
    char val )
```

Create a 32-bit union variable with the type `char`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.10.1.7 gcu_type32_f32()

```
GCU_Type32_Union gcu_type32_f32 (  
    GCU_float32_t val )
```


Create a 32-bit union variable with the type float with 32 bits.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.10.1.8 gcu_type32_i16()

```
GCU_Type32_Union gcu_type32_i16 (  
    int16_t val )
```

Create a 32-bit union variable with the type `int16_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.10.1.9 gcu_type32_i32()

```
GCU_Type32_Union gcu_type32_i32 (  
    int32_t val )
```

Create a 32-bit union variable with the type `int32_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.10.1.10 gcu_type32_i8()

```
GCU_Type32_Union gcu_type32_i8 (  
    int8_t val )
```

Create a 32-bit union variable with the type `int8_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.10.1.11 gcu_type32_ui16()

```
GCU_Type32_Union gcu_type32_ui16 (  
    uint16_t val )
```

Create a 32-bit union variable with the type `uint16_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.10.1.12 gcu_type32_ui32()

```
GCU_Type32_Union gcu_type32_ui32 (  
    uint32_t val )
```

Create a 32-bit union variable with the type `uint32_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.10.1.13 gcu_type32_ui8()

```
GCU_Type32_Union gcu_type32_ui8 (  
    uint8_t val )
```

Create a 32-bit union variable with the type `uint8_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.10.1.14 gcu_type64_c()

```
GCU_Type64_Union gcu_type64_c (  
    char val )
```

Create a 64-bit union variable with the type `char`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.10.1.15 gcu_type64_f32()

```
GCU_Type64_Union gcu_type64_f32 (  
    GCU_float32_t val )
```

Create a 64-bit union variable with the type float with 32 bits.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.10.1.16 gcu_type64_f64()

```
GCU_Type64_Union gcu_type64_f64 (  
    GCU_float64_t val )
```

Create a 64-bit union variable with the type float with 64 bits.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.10.1.17 gcu_type64_i16()

```
GCU_Type64_Union gcu_type64_i16 (
    int16_t val )
```

Create a 64-bit union variable with the type `int16_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.10.1.18 gcu_type64_i32()

```
GCU_Type64_Union gcu_type64_i32 (
    int32_t val )
```

Create a 64-bit union variable with the type `int32_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.10.1.19 gcu_type64_i64()

```
GCU_Type64_Union gcu_type64_i64 (
    int64_t val )
```


Create a 64-bit union variable with the type `int64_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.10.1.20 gcu_type64_i8()

```
GCU_Type64_Union gcu_type64_i8 (  
    int8_t val )
```

Create a 64-bit union variable with the type `int8_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.10.1.21 gcu_type64_p()

```
GCU_Type64_Union gcu_type64_p (  
    void * val )
```

Create a 64-bit union variable with the type `void *`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.10.1.22 gcu_type64_ui16()

```
GCU_Type64_Union gcu_type64_ui16 (  
    uint16_t val )
```

Create a 64-bit union variable with the type `uint16_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.10.1.23 gcu_type64_ui32()

```
GCU_Type64_Union gcu_type64_ui32 (  
    uint32_t val )
```

Create a 64-bit union variable with the type `uint32_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.10.1.24 gcu_type64_ui64()

```
GCU_Type64_Union gcu_type64_ui64 (  
    uint64_t val )
```

Create a 64-bit union variable with the type `uint64_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.10.1.25 gcu_type64_ui8()

```
GCU_Type64_Union gcu_type64_ui8 (  
    uint8_t val )
```

Create a 64-bit union variable with the type `uint8_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.10.1.26 gcu_type8_c()

```
GCU_Type8_Union gcu_type8_c (  
    char val )
```

Create a 8-bit union variable with the type `char`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.10.1.27 gcu_type8_i8()

```
GCU_Type8_Union gcu_type8_i8 (  
    int8_t val )
```

Create a 8-bit union variable with the type `int8_t`.

Parameters

<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.10.1.28 gcu_type8_ui8()

```
GCU_Type8_Union gcu_type8_ui8 (  
    uint8_t val )
```

Create a 8-bit union variable with the type `uint8_t`.

Parameters

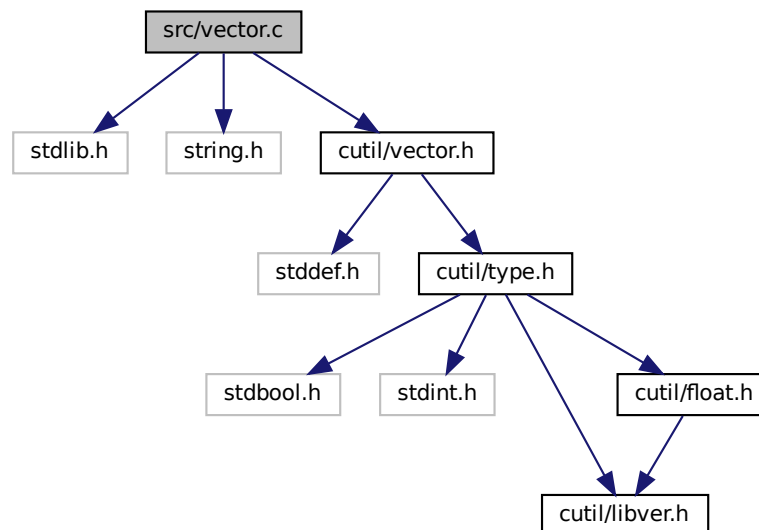
<i>val</i>	The value to put into the union.
------------	----------------------------------

Returns

The union variable.

5.11 src/vector.c File Reference

```
#include <stdlib.h>
#include <string.h>
#include "cutil/vector.h"
Include dependency graph for vector.c:
```



Macros

- `#define GROWTH_FACTOR 1.3`

Functions

- `GCU_Vector64 * gcu_vector64_create (size_t count)`
Create a vector structure.
- `void gcu_vector64_destroy (GCU_Vector64 *vector)`
Destroy a vector structure and clean up memory allocations.
- `bool gcu_vector64_append (GCU_Vector64 *vector, GCU_Type64_Union value)`
Append an item at the end of the vector.

- `size_t gcu_vector64_count (GCU_Vector64 *vector)`
Get a count of entries in the vector.
- `GCU_Vector32 * gcu_vector32_create (size_t count)`
Create a vector structure.
- `void gcu_vector32_destroy (GCU_Vector32 *vector)`
Destroy a vector structure and clean up memory allocations.
- `bool gcu_vector32_append (GCU_Vector32 *vector, GCU_Type32_Union value)`
Append an item at the end of the vector.
- `size_t gcu_vector32_count (GCU_Vector32 *vector)`
Get a count of entries in the vector.
- `GCU_Vector16 * gcu_vector16_create (size_t count)`
Create a vector structure.
- `void gcu_vector16_destroy (GCU_Vector16 *vector)`
Destroy a vector structure and clean up memory allocations.
- `bool gcu_vector16_append (GCU_Vector16 *vector, GCU_Type16_Union value)`
Append an item at the end of the vector.
- `size_t gcu_vector16_count (GCU_Vector16 *vector)`
Get a count of entries in the vector.
- `GCU_Vector8 * gcu_vector8_create (size_t count)`
Create a vector structure.
- `void gcu_vector8_destroy (GCU_Vector8 *vector)`
Destroy a vector structure and clean up memory allocations.
- `bool gcu_vector8_append (GCU_Vector8 *vector, GCU_Type8_Union value)`
Append an item at the end of the vector.
- `size_t gcu_vector8_count (GCU_Vector8 *vector)`
Get a count of entries in the vector.

5.11.1 Function Documentation

5.11.1.1 gcu_vector16_append()

```
bool gcu_vector16_append (
    GCU_Vector16 * vector,
    GCU_Type16_Union value )
```

Append an item at the end of the vector.

If there is not enough space in the current data structure, new space will be attempted to be allocated. This may invalidate any pointers to the previous data locations.

Parameters

<i>value</i>	The item to append to the end of the vector.
--------------	--

Returns

`true` on success, `false` otherwise.

5.11.1.2 gcu_vector16_count()

```
size_t gcu_vector16_count (
    GCU_Vector16 * vector )
```

Get a count of entries in the vector.

Parameters

<i>vector</i>	The vector structure on which to operate.
---------------	---

Returns

The count of entries in the vector.

5.11.1.3 gcu_vector16_create()

```
GCU_Vector16* gcu_vector16_create (
    size_t count )
```

Create a vector structure.

All invocations of a vector must have a corresponding [gcu_vector16_destroy\(\)](#) call in order to clean up dynamically-allocated memory.

The vector 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 vector.

Parameters

<i>count</i>	The number of items anticipated to be stored in the vector.
--------------	---

Returns

A struct containing the vector information.

5.11.1.4 gcu_vector16_destroy()

```
void gcu_vector16_destroy (
    GCU_Vector16 * vector )
```

Destroy a vector 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

<i>vector</i>	The vector structure to be destroyed.
---------------	---------------------------------------

5.11.1.5 gcu_vector32_append()

```
bool gcu_vector32_append (
    GCU_Vector32 * vector,
    GCU_Type32_Union value )
```

Append an item at the end of the vector.

If there is not enough space in the current data structure, new space will be attempted to be allocated. This may invalidate any pointers to the previous data locations.

Parameters

<i>value</i>	The item to append to the end of the vector.
--------------	--

Returns

`true` on success, `false` otherwise.

5.11.1.6 gcu_vector32_count()

```
size_t gcu_vector32_count (
    GCU_Vector32 * vector )
```

Get a count of entries in the vector.

Parameters

<i>vector</i>	The vector structure on which to operate.
---------------	---

Returns

The count of entries in the vector.

5.11.1.7 gcu_vector32_create()

```
GCU_Vector32* gcu_vector32_create (
    size_t count )
```

Create a vector structure.

All invocations of a vector must have a corresponding `gcu_vector32_destroy()` call in order to clean up dynamically-allocated memory.

The vector 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 vector.

Parameters

<i>count</i>	The number of items anticipated to be stored in the vector.
--------------	---

Returns

A struct containing the vector information.

5.11.1.8 gcu_vector32_destroy()

```
void gcu_vector32_destroy (
    GCU_Vector32 * vector )
```

Destroy a vector 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

<i>vector</i>	The vector structure to be destroyed.
---------------	---------------------------------------

5.11.1.9 gcu_vector64_append()

```
bool gcu_vector64_append (
    GCU_Vector64 * vector,
    GCU_Type64_Union value )
```

Append an item at the end of the vector.

If there is not enough space in the current data structure, new space will be attempted to be allocated. This may invalidate any pointers to the previous data locations.

Parameters

<i>value</i>	The item to append to the end of the vector.
--------------	--

Returns

`true` on success, `false` otherwise.

5.11.1.10 `gcu_vector64_count()`

```
size_t gcu_vector64_count (
    GCU_Vector64 * vector )
```

Get a count of entries in the vector.

Parameters

<code>vector</code>	The vector structure on which to operate.
---------------------	---

Returns

The count of entries in the vector.

5.11.1.11 `gcu_vector64_create()`

```
GCU_Vector64* gcu_vector64_create (
    size_t count )
```

Create a vector structure.

All invocations of a vector must have a corresponding `gcu_vector64_destroy()` call in order to clean up dynamically-allocated memory.

The vector 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 vector.

Parameters

<code>count</code>	The number of items anticipated to be stored in the vector.
--------------------	---

Returns

A struct containing the vector information.

5.11.1.12 `gcu_vector64_destroy()`

```
void gcu_vector64_destroy (
    GCU_Vector64 * vector )
```

Destroy a vector 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

<i>vector</i>	The vector structure to be destroyed.
---------------	---------------------------------------

5.11.1.13 `gcu_vector8_append()`

```
bool gcu_vector8_append (
    GCU_Vector8 * vector,
    GCU_Type8_Union value )
```

Append an item at the end of the vector.

If there is not enough space in the current data structure, new space will be attempted to be allocated. This may invalidate any pointers to the previous data locations.

Parameters

<i>value</i>	The item to append to the end of the vector.
--------------	--

Returns

`true` on success, `false` otherwise.

5.11.1.14 `gcu_vector8_count()`

```
size_t gcu_vector8_count (
    GCU_Vector8 * vector )
```

Get a count of entries in the vector.

Parameters

<i>vector</i>	The vector structure on which to operate.
---------------	---

Returns

The count of entries in the vector.

5.11.1.15 `gcu_vector8_create()`

```
GCU_Vector8* gcu_vector8_create (
    size_t count )
```

Create a vector structure.

All invocations of a vector must have a corresponding `gcu_vector8_destroy()` call in order to clean up dynamically-allocated memory.

The vector 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 vector.

Parameters

<i>count</i>	The number of items anticipated to be stored in the vector.
--------------	---

Returns

A struct containing the vector information.

5.11.1.16 `gcu_vector8_destroy()`

```
void gcu_vector8_destroy (
    GCU_Vector8 * vector )
```

Destroy a vector 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.

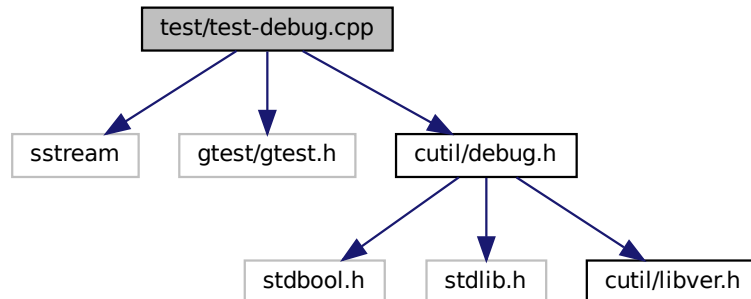
Parameters

<i>vector</i>	The vector structure to be destroyed.
---------------	---------------------------------------

5.12 `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`)

5.12.1 Detailed Description

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

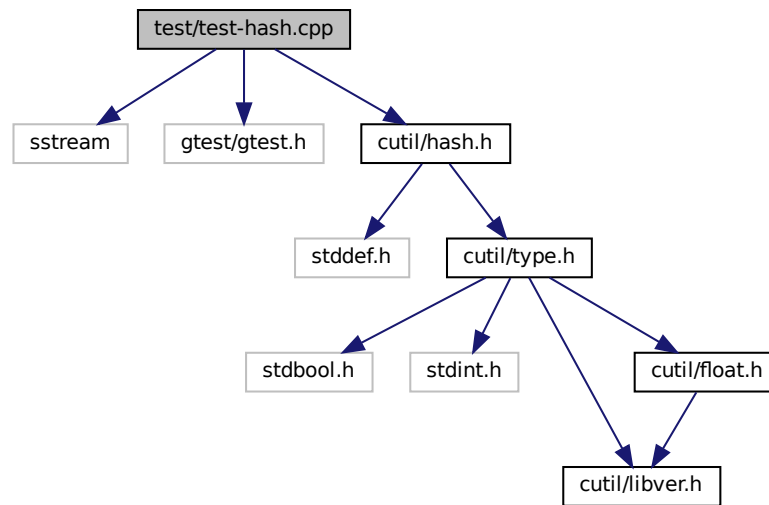
5.13 test/test-hash.cpp File Reference

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** (Hash64, CreateEmpty)
- **TEST** (Hash64, Create)
- **TEST** (Hash64, Set)
- **TEST** (Hash64, Remove)
- **TEST** (Hash64, IteratorOnEmpty)
- **TEST** (Hash64, Iterator)
- **TEST** (Hash32, CreateEmpty)
- **TEST** (Hash32, Create)
- **TEST** (Hash32, Set)
- **TEST** (Hash32, Remove)
- **TEST** (Hash32, IteratorOnEmpty)
- **TEST** (Hash32, Iterator)
- **TEST** (Hash16, CreateEmpty)
- **TEST** (Hash16, Create)
- **TEST** (Hash16, Set)
- **TEST** (Hash16, Remove)
- **TEST** (Hash16, IteratorOnEmpty)
- **TEST** (Hash16, Iterator)
- **TEST** (Hash8, CreateEmpty)
- **TEST** (Hash8, Create)
- **TEST** (Hash8, Set)
- **TEST** (Hash8, Remove)
- **TEST** (Hash8, IteratorOnEmpty)
- **TEST** (Hash8, Iterator)
- int **main** (int argc, char **argv)

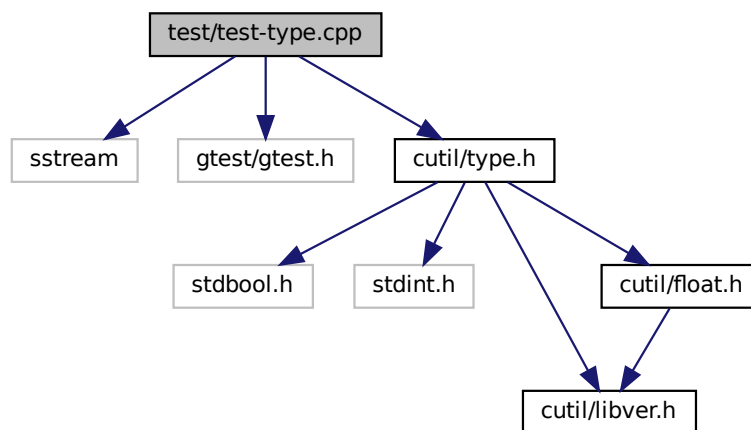
5.13.1 Detailed Description

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

5.14 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

- **TEST** (Type, Union)
- `int main` (int argc, char **argv)

5.14.1 Detailed Description

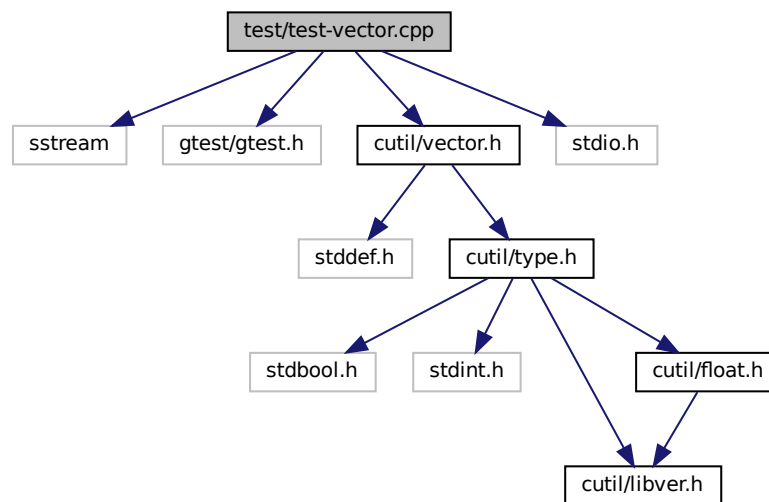
Test the behavior of Ghoti.io CUtil type library.

5.15 test/test-vector.cpp File Reference

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

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

Include dependency graph for test-vector.cpp:



Functions

- **TEST** (Vector64, CreateEmpty)
- **TEST** (Vector64, NonEmpty)
- **TEST** (Vector32, CreateEmpty)
- **TEST** (Vector32, NonEmpty)
- **TEST** (Vector16, CreateEmpty)
- **TEST** (Vector16, NonEmpty)
- **TEST** (Vector8, CreateEmpty)
- **TEST** (Vector8, NonEmpty)
- **int main** (int argc, char **argv)

5.15.1 Detailed Description

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

Index

- debug.c
 - gcu_calloc, [85](#)
 - gcu_free, [85](#)
 - gcu_malloc, [85](#)
 - gcu_realloc, [86](#)
- debug.h
 - gcu_calloc, [30](#)
 - gcu_free, [31](#)
 - gcu_malloc, [31](#)
 - gcu_realloc, [31](#)
- gcu_calloc
 - debug.c, [85](#)
 - debug.h, [30](#)
- gcu_free
 - debug.c, [85](#)
 - debug.h, [31](#)
- GCU_Hash16_Cell, [7](#)
- gcu_hash16_contains
 - hash.c, [90](#)
 - hash.h, [36](#)
- gcu_hash16_count
 - hash.c, [90](#)
 - hash.h, [36](#)
- gcu_hash16_create
 - hash.c, [90](#)
 - hash.h, [37](#)
- gcu_hash16_destroy
 - hash.c, [91](#)
 - hash.h, [37](#)
- gcu_hash16_get
 - hash.c, [91](#)
 - hash.h, [37](#)
- GCU_Hash16_Iterator, [8](#)
- gcu_hash16_iterator_get
 - hash.c, [92](#)
 - hash.h, [38](#)
- gcu_hash16_iterator_next
 - hash.c, [92](#)
 - hash.h, [38](#)
- gcu_hash16_remove
 - hash.c, [93](#)
 - hash.h, [39](#)
- gcu_hash16_set
 - hash.c, [93](#)
 - hash.h, [39](#)
- GCU_Hash16_Table, [9](#)
- GCU_Hash16_Value, [10](#)
- GCU_Hash32_Cell, [11](#)
- gcu_hash32_contains
 - hash.c, [94](#)
 - hash.h, [40](#)
- gcu_hash32_count
 - hash.c, [94](#)
 - hash.h, [40](#)
- gcu_hash32_create
 - hash.c, [95](#)
 - hash.h, [41](#)
- gcu_hash32_destroy
 - hash.c, [95](#)
 - hash.h, [41](#)
- gcu_hash32_get
 - hash.c, [95](#)
 - hash.h, [42](#)
- GCU_Hash32_Iterator, [12](#)
- gcu_hash32_iterator_get
 - hash.c, [96](#)
 - hash.h, [42](#)
- gcu_hash32_iterator_next
 - hash.c, [96](#)
 - hash.h, [42](#)
- gcu_hash32_remove
 - hash.c, [97](#)
 - hash.h, [43](#)
- gcu_hash32_set
 - hash.c, [97](#)
 - hash.h, [44](#)
- GCU_Hash32_Table, [13](#)
- GCU_Hash32_Value, [14](#)
- GCU_Hash64_Cell, [15](#)
- gcu_hash64_contains
 - hash.c, [98](#)
 - hash.h, [44](#)
- gcu_hash64_count
 - hash.c, [98](#)
 - hash.h, [44](#)
- gcu_hash64_create
 - hash.c, [99](#)
 - hash.h, [45](#)
- gcu_hash64_destroy
 - hash.c, [99](#)
 - hash.h, [45](#)
- gcu_hash64_get
 - hash.c, [100](#)
 - hash.h, [46](#)
- GCU_Hash64_Iterator, [16](#)
- gcu_hash64_iterator_get
 - hash.c, [100](#)
 - hash.h, [46](#)

- gcu_hash64_iterator_next
 - hash.c, [100](#)
 - hash.h, [47](#)
- gcu_hash64_remove
 - hash.c, [101](#)
 - hash.h, [47](#)
- gcu_hash64_set
 - hash.c, [102](#)
 - hash.h, [48](#)
- GCU_Hash64_Table, [17](#)
- GCU_Hash64_Value, [18](#)
- GCU_Hash8_Cell, [19](#)
- gcu_hash8_contains
 - hash.c, [102](#)
 - hash.h, [48](#)
- gcu_hash8_count
 - hash.c, [102](#)
 - hash.h, [50](#)
- gcu_hash8_create
 - hash.c, [103](#)
 - hash.h, [50](#)
- gcu_hash8_destroy
 - hash.c, [103](#)
 - hash.h, [51](#)
- gcu_hash8_get
 - hash.c, [104](#)
 - hash.h, [51](#)
- GCU_Hash8_Iterator, [20](#)
- gcu_hash8_iterator_get
 - hash.c, [104](#)
 - hash.h, [51](#)
- gcu_hash8_iterator_next
 - hash.c, [105](#)
 - hash.h, [52](#)
- gcu_hash8_remove
 - hash.c, [105](#)
 - hash.h, [53](#)
- gcu_hash8_set
 - hash.c, [106](#)
 - hash.h, [53](#)
- GCU_Hash8_Table, [21](#)
- GCU_Hash8_Value, [22](#)
- gcu_malloc
 - debug.c, [85](#)
 - debug.h, [31](#)
- gcu_realloc
 - debug.c, [86](#)
 - debug.h, [31](#)
- gcu_type16_c
 - type.c, [108](#)
 - type.h, [58](#)
- gcu_type16_i16
 - type.c, [109](#)
 - type.h, [58](#)
- gcu_type16_i8
 - type.c, [109](#)
 - type.h, [59](#)
- gcu_type16_ui16
 - type.c, [109](#)
 - type.h, [59](#)
- gcu_type16_ui8
 - type.c, [110](#)
 - type.h, [60](#)
- GCU_Type16_Union, [23](#)
- gcu_type32_c
 - type.c, [110](#)
 - type.h, [60](#)
- gcu_type32_f32
 - type.c, [110](#)
 - type.h, [60](#)
- gcu_type32_i16
 - type.c, [112](#)
 - type.h, [61](#)
- gcu_type32_i32
 - type.c, [112](#)
 - type.h, [61](#)
- gcu_type32_i8
 - type.c, [112](#)
 - type.h, [61](#)
- gcu_type32_ui16
 - type.c, [114](#)
 - type.h, [62](#)
- gcu_type32_ui32
 - type.c, [114](#)
 - type.h, [62](#)
- gcu_type32_ui64
 - type.h, [62](#)
- gcu_type32_ui8
 - type.c, [114](#)
 - type.h, [64](#)
- GCU_Type32_Union, [23](#)
- gcu_type64_c
 - type.c, [116](#)
 - type.h, [64](#)
- gcu_type64_f32
 - type.c, [116](#)
 - type.h, [64](#)
- gcu_type64_f64
 - type.c, [116](#)
 - type.h, [66](#)
- gcu_type64_i16
 - type.c, [118](#)
 - type.h, [66](#)
- gcu_type64_i32
 - type.c, [118](#)
 - type.h, [66](#)
- gcu_type64_i64
 - type.c, [118](#)
 - type.h, [68](#)
- gcu_type64_i8
 - type.c, [120](#)
 - type.h, [68](#)
- gcu_type64_p
 - type.c, [120](#)
 - type.h, [68](#)
- gcu_type64_ui16

- type.c, [120](#)
- type.h, [70](#)
- gcu_type64_ui32
 - type.c, [122](#)
 - type.h, [70](#)
- gcu_type64_ui64
 - type.c, [122](#)
 - type.h, [70](#)
- gcu_type64_ui8
 - type.c, [122](#)
 - type.h, [72](#)
- GCU_Type64_Union, [24](#)
- gcu_type8_c
 - type.c, [124](#)
 - type.h, [72](#)
- gcu_type8_i8
 - type.c, [124](#)
 - type.h, [72](#)
- gcu_type8_ui8
 - type.c, [124](#)
 - type.h, [74](#)
- GCU_Type8_Union, [24](#)
- GCU_Vector16, [25](#)
- gcu_vector16_append
 - vector.c, [127](#)
 - vector.h, [76](#)
- gcu_vector16_count
 - vector.c, [128](#)
 - vector.h, [77](#)
- gcu_vector16_create
 - vector.c, [128](#)
 - vector.h, [77](#)
- gcu_vector16_destroy
 - vector.c, [128](#)
 - vector.h, [78](#)
- GCU_Vector32, [26](#)
- gcu_vector32_append
 - vector.c, [130](#)
 - vector.h, [78](#)
- gcu_vector32_count
 - vector.c, [130](#)
 - vector.h, [78](#)
- gcu_vector32_create
 - vector.c, [130](#)
 - vector.h, [79](#)
- gcu_vector32_destroy
 - vector.c, [131](#)
 - vector.h, [79](#)
- GCU_Vector64, [27](#)
- gcu_vector64_append
 - vector.c, [131](#)
 - vector.h, [80](#)
- gcu_vector64_count
 - vector.c, [132](#)
 - vector.h, [80](#)
- gcu_vector64_create
 - vector.c, [132](#)
 - vector.h, [80](#)
- gcu_vector64_destroy
 - vector.c, [132](#)
 - vector.h, [82](#)
- GCU_Vector8, [28](#)
- gcu_vector8_append
 - vector.c, [133](#)
 - vector.h, [82](#)
- gcu_vector8_count
 - vector.c, [133](#)
 - vector.h, [82](#)
- gcu_vector8_create
 - vector.c, [134](#)
 - vector.h, [83](#)
- gcu_vector8_destroy
 - vector.c, [134](#)
 - vector.h, [83](#)
- GHOTIIO_CUTIL
 - libver.h, [54](#)
- GHOTIIO_CUTIL_CONCAT
 - libver.h, [55](#)
- GHOTIIO_CUTIL_CONCAT_INNER
 - libver.h, [55](#)
- GHOTIIO_CUTIL_NAME
 - libver.h, [55](#)
- hash.c
 - gcu_hash16_contains, [90](#)
 - gcu_hash16_count, [90](#)
 - gcu_hash16_create, [90](#)
 - gcu_hash16_destroy, [91](#)
 - gcu_hash16_get, [91](#)
 - gcu_hash16_iterator_get, [92](#)
 - gcu_hash16_iterator_next, [92](#)
 - gcu_hash16_remove, [93](#)
 - gcu_hash16_set, [93](#)
 - gcu_hash32_contains, [94](#)
 - gcu_hash32_count, [94](#)
 - gcu_hash32_create, [95](#)
 - gcu_hash32_destroy, [95](#)
 - gcu_hash32_get, [95](#)
 - gcu_hash32_iterator_get, [96](#)
 - gcu_hash32_iterator_next, [96](#)
 - gcu_hash32_remove, [97](#)
 - gcu_hash32_set, [97](#)
 - gcu_hash64_contains, [98](#)
 - gcu_hash64_count, [98](#)
 - gcu_hash64_create, [99](#)
 - gcu_hash64_destroy, [99](#)
 - gcu_hash64_get, [100](#)
 - gcu_hash64_iterator_get, [100](#)
 - gcu_hash64_iterator_next, [100](#)
 - gcu_hash64_remove, [101](#)
 - gcu_hash64_set, [102](#)
 - gcu_hash8_contains, [102](#)
 - gcu_hash8_count, [102](#)
 - gcu_hash8_create, [103](#)
 - gcu_hash8_destroy, [103](#)
 - gcu_hash8_get, [104](#)
 - gcu_hash8_iterator_get, [104](#)

- gcu_hash8_iterator_next, 105
- gcu_hash8_remove, 105
- gcu_hash8_set, 106
- hash.h
 - gcu_hash16_contains, 36
 - gcu_hash16_count, 36
 - gcu_hash16_create, 37
 - gcu_hash16_destroy, 37
 - gcu_hash16_get, 37
 - gcu_hash16_iterator_get, 38
 - gcu_hash16_iterator_next, 38
 - gcu_hash16_remove, 39
 - gcu_hash16_set, 39
 - gcu_hash32_contains, 40
 - gcu_hash32_count, 40
 - gcu_hash32_create, 41
 - gcu_hash32_destroy, 41
 - gcu_hash32_get, 42
 - gcu_hash32_iterator_get, 42
 - gcu_hash32_iterator_next, 42
 - gcu_hash32_remove, 43
 - gcu_hash32_set, 44
 - gcu_hash64_contains, 44
 - gcu_hash64_count, 44
 - gcu_hash64_create, 45
 - gcu_hash64_destroy, 45
 - gcu_hash64_get, 46
 - gcu_hash64_iterator_get, 46
 - gcu_hash64_iterator_next, 47
 - gcu_hash64_remove, 47
 - gcu_hash64_set, 48
 - gcu_hash8_contains, 48
 - gcu_hash8_count, 50
 - gcu_hash8_create, 50
 - gcu_hash8_destroy, 51
 - gcu_hash8_get, 51
 - gcu_hash8_iterator_get, 51
 - gcu_hash8_iterator_next, 52
 - gcu_hash8_remove, 53
 - gcu_hash8_set, 53
- include/cutil/debug.h, 29
- include/cutil/float.h, 32
- include/cutil/hash.h, 33
- include/cutil/libver.h, 53
- include/cutil/type.h, 56
- include/cutil/vector.h, 74
- libver.h
 - GHOTIO_CUTIL, 54
 - GHOTIO_CUTIL_CONCAT, 55
 - GHOTIO_CUTIL_CONCAT_INNER, 55
 - GHOTIO_CUTIL_NAME, 55
- src/debug.c, 84
- src/float_identifier.c, 86
- src/hash.c, 87
- src/type.c, 106
- src/vector.c, 126
- test/test-debug.cpp, 134
- test/test-hash.cpp, 135
- test/test-type.cpp, 137
- test/test-vector.cpp, 138
- type.c
 - gcu_type16_c, 108
 - gcu_type16_i16, 109
 - gcu_type16_i8, 109
 - gcu_type16_ui16, 109
 - gcu_type16_ui8, 110
 - gcu_type32_c, 110
 - gcu_type32_f32, 110
 - gcu_type32_i16, 112
 - gcu_type32_i32, 112
 - gcu_type32_i8, 112
 - gcu_type32_ui16, 114
 - gcu_type32_ui32, 114
 - gcu_type32_ui8, 114
 - gcu_type64_c, 116
 - gcu_type64_f32, 116
 - gcu_type64_f64, 116
 - gcu_type64_i16, 118
 - gcu_type64_i32, 118
 - gcu_type64_i64, 118
 - gcu_type64_i8, 120
 - gcu_type64_p, 120
 - gcu_type64_ui16, 120
 - gcu_type64_ui32, 122
 - gcu_type64_ui64, 122
 - gcu_type64_ui8, 122
 - gcu_type8_c, 124
 - gcu_type8_i8, 124
 - gcu_type8_ui8, 124
- type.h
 - gcu_type16_c, 58
 - gcu_type16_i16, 58
 - gcu_type16_i8, 59
 - gcu_type16_ui16, 59
 - gcu_type16_ui8, 60
 - gcu_type32_c, 60
 - gcu_type32_f32, 60
 - gcu_type32_i16, 61
 - gcu_type32_i32, 61
 - gcu_type32_i8, 61
 - gcu_type32_ui16, 62
 - gcu_type32_ui32, 62
 - gcu_type32_ui64, 62
 - gcu_type32_ui8, 64
 - gcu_type64_c, 64
 - gcu_type64_f32, 64
 - gcu_type64_f64, 66
 - gcu_type64_i16, 66
 - gcu_type64_i32, 66
 - gcu_type64_i64, 68
 - gcu_type64_i8, 68
 - gcu_type64_p, 68
 - gcu_type64_ui16, 70
 - gcu_type64_ui32, 70

- [gcu_type64_ui64, 70](#)
- [gcu_type64_ui8, 72](#)
- [gcu_type8_c, 72](#)
- [gcu_type8_i8, 72](#)
- [gcu_type8_ui8, 74](#)

vector.c

- [gcu_vector16_append, 127](#)
- [gcu_vector16_count, 128](#)
- [gcu_vector16_create, 128](#)
- [gcu_vector16_destroy, 128](#)
- [gcu_vector32_append, 130](#)
- [gcu_vector32_count, 130](#)
- [gcu_vector32_create, 130](#)
- [gcu_vector32_destroy, 131](#)
- [gcu_vector64_append, 131](#)
- [gcu_vector64_count, 132](#)
- [gcu_vector64_create, 132](#)
- [gcu_vector64_destroy, 132](#)
- [gcu_vector8_append, 133](#)
- [gcu_vector8_count, 133](#)
- [gcu_vector8_create, 134](#)
- [gcu_vector8_destroy, 134](#)

vector.h

- [gcu_vector16_append, 76](#)
- [gcu_vector16_count, 77](#)
- [gcu_vector16_create, 77](#)
- [gcu_vector16_destroy, 78](#)
- [gcu_vector32_append, 78](#)
- [gcu_vector32_count, 78](#)
- [gcu_vector32_create, 79](#)
- [gcu_vector32_destroy, 79](#)
- [gcu_vector64_append, 80](#)
- [gcu_vector64_count, 80](#)
- [gcu_vector64_create, 80](#)
- [gcu_vector64_destroy, 82](#)
- [gcu_vector8_append, 82](#)
- [gcu_vector8_count, 82](#)
- [gcu_vector8_create, 83](#)
- [gcu_vector8_destroy, 83](#)