Ghoti.io CUtil

0.1

Generated by Doxygen 1.9.1

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:

Chapter 3

File Index

3.1 File List

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

Chapter 4

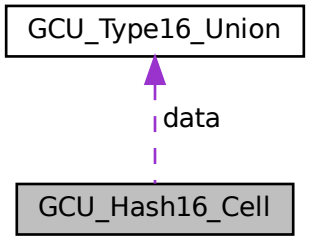
Class Documentation

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:

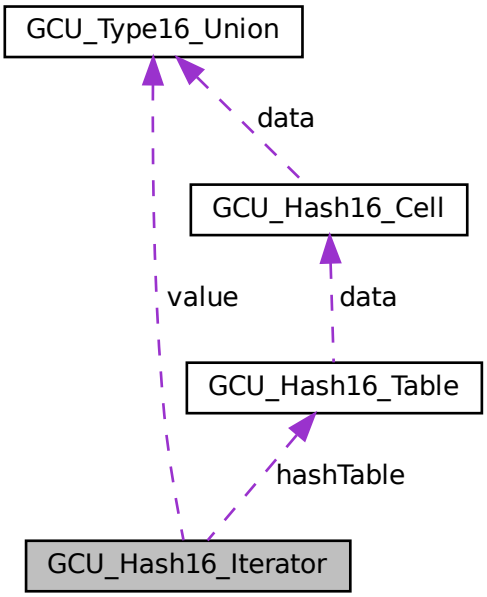
- include/cutil/hash.h

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:

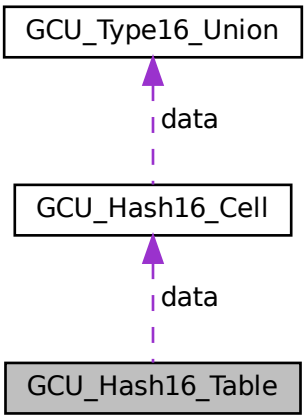
- include/cutil/hash.h

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:

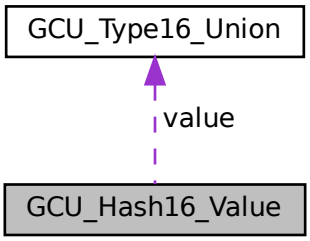
- include/cutil/hash.h

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 imperitive that the hash table be able to indicate whether or not the value existed in the table. Both goals are accomplished by this approach.

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

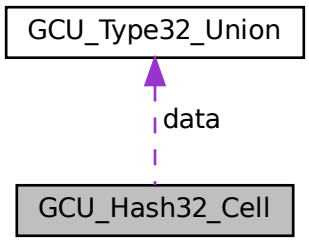
- include/cutil/hash.h

4.5 GCU_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:

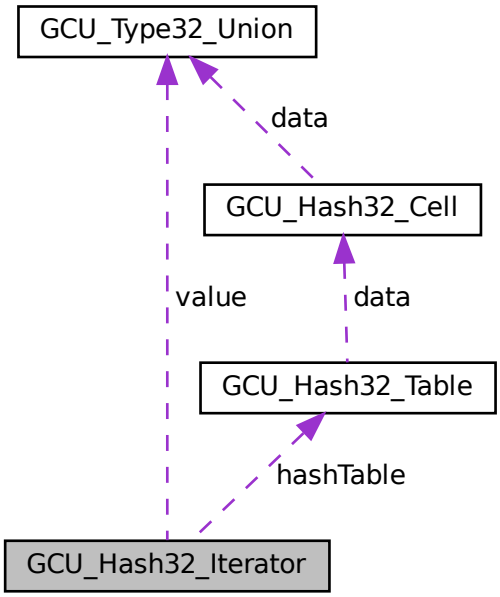
- include/cutil/hash.h

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:

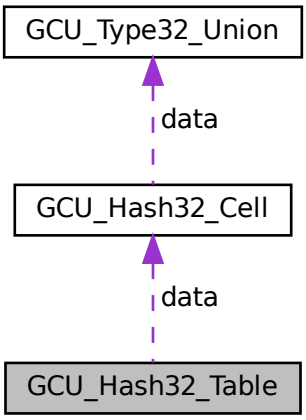
- include/cutil/hash.h

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:

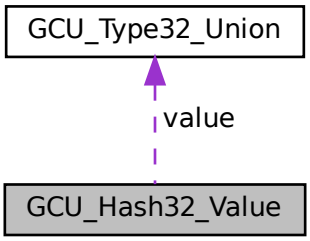
- include/cutil/hash.h

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 imperitive that the hash table be able to indicate whether or not the value existed in the table. Both goals are accomplished by this approach.

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

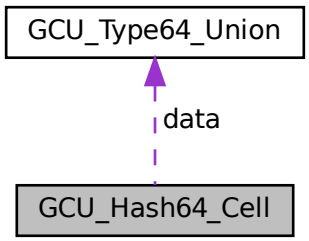
- include/cutil/hash.h

4.9 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:

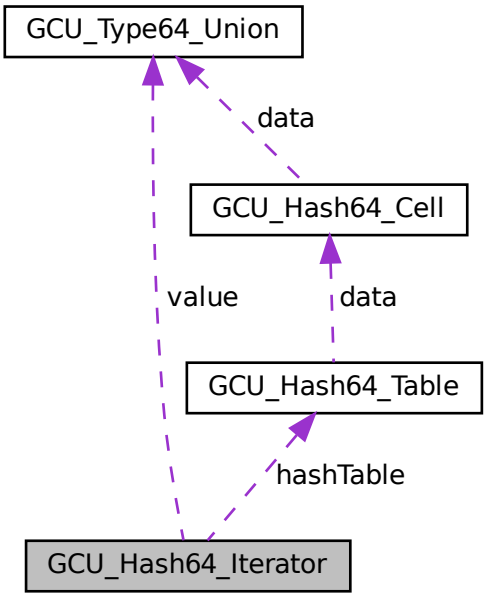
- include/cutil/hash.h

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:

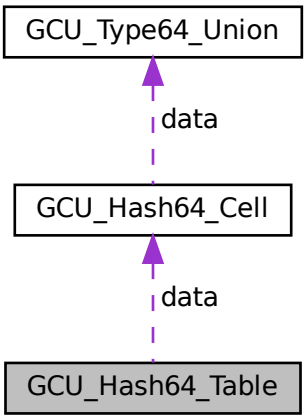
- include/cutil/hash.h

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:

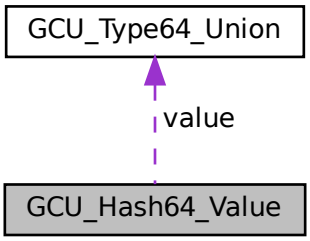
- include/cutil/hash.h

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 imperitive that the hash table be able to indicate whether or not the value existed in the table. Both goals are accomplished by this approach.

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

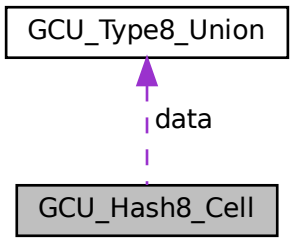
- include/cutil/hash.h

4.13 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:

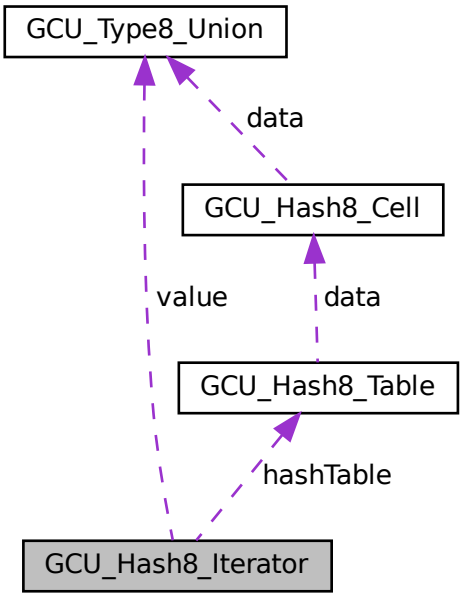
- include/cutil/hash.h

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:

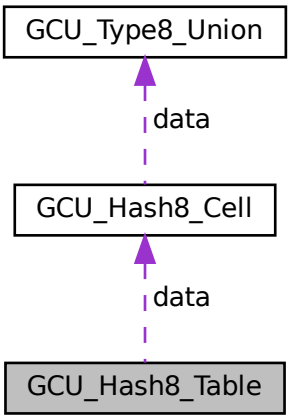
- include/cutil/hash.h

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:

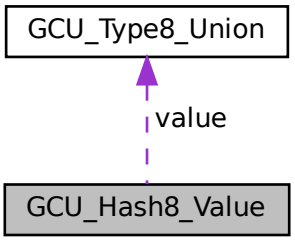
- include/cutil/hash.h

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 imperitive that the hash table be able to indicate whether or not the value existed in the table. Both goals are accomplished by this approach.

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

- include/cutil/hash.h

4.17 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
  uint16_t value.
- uint8_t ui8
  uint8_t value.
- int16_t i16
  int16_t value.
- int8_t i8
  int8_t value.
- char c
  char value.

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
  uint32_t value.
- uint16_t ui16
  uint16_t value.
- uint8_t ui8
  uint8_t value.
- int32_t i32
  int32_t value.
- int16_t i16
  int16_t value.
- int8_t i8
  int8_t value.
- GCU_float32_t f32
  32-bit float value.
- wchar_t wc
  wchar_t value.
- char c
  char value.

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
  Pointer type value.
- uint64_t ui64
  uint64_t value.
- uint32_t ui32
  uint32_t value.
- uint16_t ui16
  uint16_t value.
- uint8_t ui8
  uint8_t value.
- int64_t i64

int64_t value.

- int32_t i32

  int32_t value.
- int16_t i16

  int16_t value.
- int8_t i8

  int8_t value.
- GCU_float64_t f64

  64-bit float value.
- GCU_float32_t f32

  32-bit float value.
- wchar_t wc

  wchar_t value.
- char c

  char value.

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

  uint8_t value.
- int8_t i8

  int8_t value.
- char c

  char value.

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:

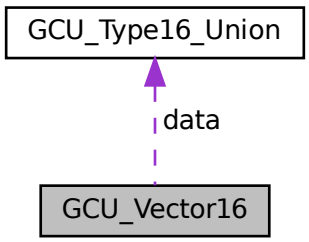
- include/cutil/type.h

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:

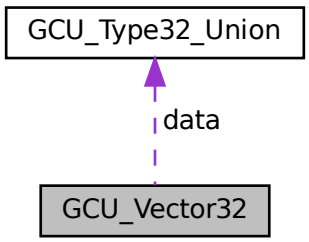
- include/cutil/vector.h

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:

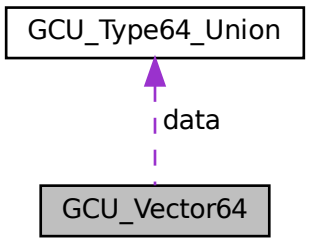
- include/cutil/vector.h

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:

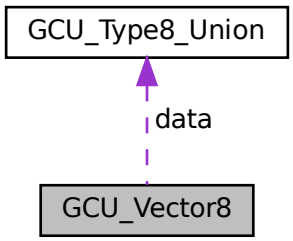
- include/cutil/vector.h

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

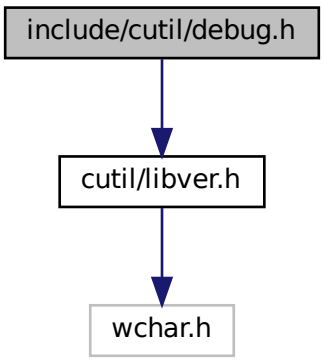
File Documentation

5.1 include/cutil/debug.h File Reference

Header file for debugging-related functions.

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

Include dependency graph for debug.h:



5.1.1 Detailed Description

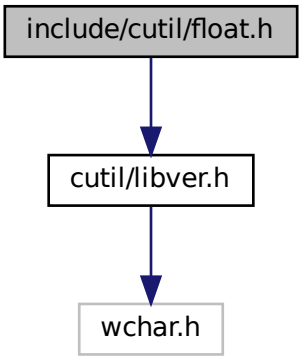
Header file for debugging-related functions.

5.2 include/cutil/float.h File Reference

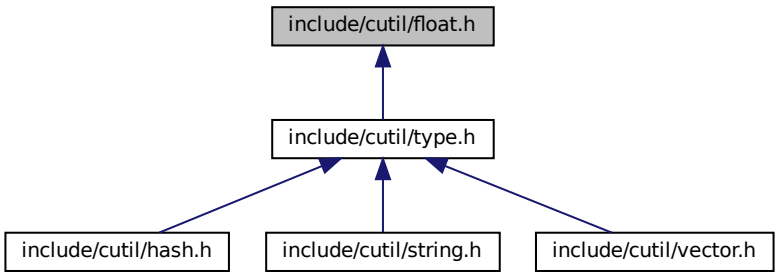
Type definitions for float types.

```
#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

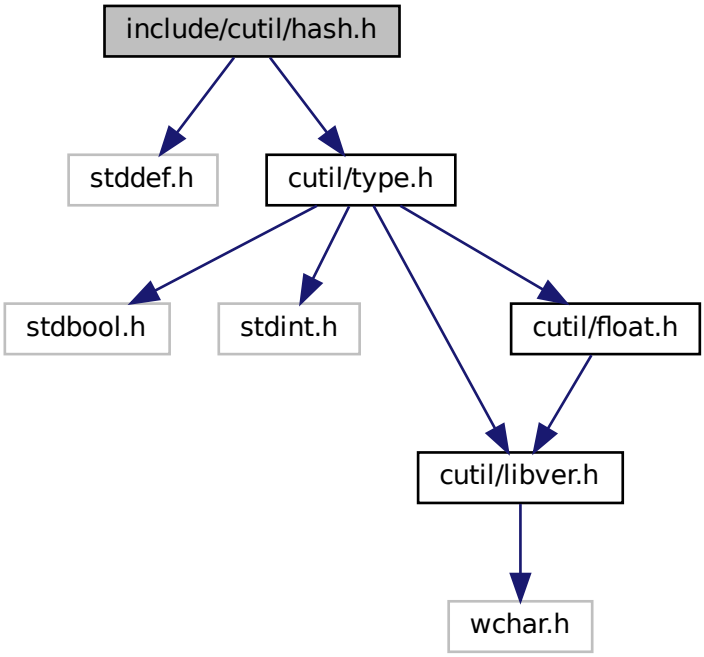
Type definitions for float types.

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:



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

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

Returns

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

5.3.2.2 gcu_hash16_count()

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

Get a count of active entries in the hash table.

Parameters

hashTable	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

count	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

hashTable	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

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

Returns

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

5.3.2.6 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

hashTable	The hash table structure on which to operate.

Returns

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

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

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

Returns

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

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

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

Returns

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

5.3.2.9 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

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

Returns

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

5.3.2.10 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

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

Returns

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

5.3.2.11 gcu_hash32_count()

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

Get a count of active entries in the hash table.

Parameters

hashTable	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

count	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

hashTable	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

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

Returns

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

5.3.2.15 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

hashTable	The hash table structure on which to operate.

Returns

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

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

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

Returns

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

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

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

Returns

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

5.3.2.18 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

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

Returns

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

5.3.2.19 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

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

Returns

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

5.3.2.20 gcu_hash64_count()

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

Get a count of active entries in the hash table.

Parameters

hashTable	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

count	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

hashTable	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

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

Returns

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

5.3.2.24 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

hashTable	The hash table structure on which to operate.

Returns

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

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

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

Returns

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

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

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

Returns

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

5.3.2.27 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

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

Returns

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

5.3.2.28 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

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

Returns

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

5.3.2.29 gcu_hash8_count()

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

Get a count of active entries in the hash table.

Parameters

hashTable	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

count	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

hashTable	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

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

Returns

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

5.3.2.33 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

hashTable	The hash table structure on which to operate.

Returns

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

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

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

Returns

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

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

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

Returns

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

5.3.2.36 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

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

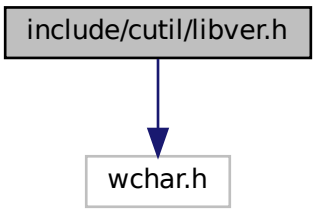
Returns

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

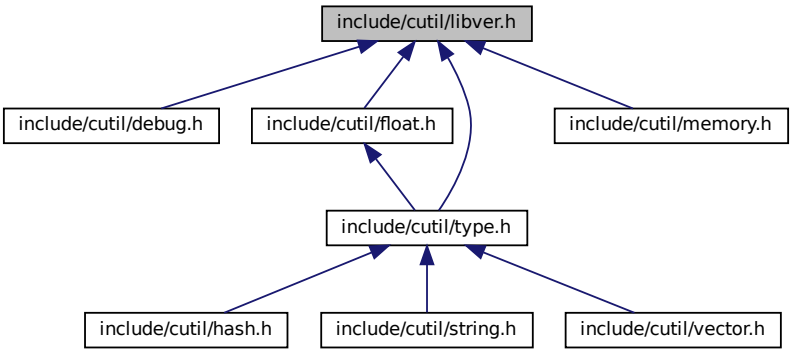
5.4 include/cutil/libver.h File Reference

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

`#include <wchar.h>`
Include dependency graph for libver.h:



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_RENAME(GHOTIIO_CUTIL_NAME, _ ## NAME)

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

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

  Helper macro to concatenate the `#define`s properly.
- #define GHOTIIO_CUTIL_CONCAT2_INNER(a, b) a ## b

  Helper macro to concatenate the identifiers.
- #define GHOTIIO_CUTIL_CONCAT2(a, b) GHOTIIO_CUTIL_CONCAT2_INNER(a,b)

  Helper macro to concatenate the identifiers.
- #define GHOTIIO_CUTIL_CONCAT3_INNER(a, b, c) a ## b ## c

  Helper macro to concatenate the identifiers.
- #define GHOTIIO_CUTIL_CONCAT3(a, b, c) GHOTIIO_CUTIL_CONCAT3_INNER(a,b,c)

  Helper macro to concatenate the identifiers.
- #define GCU_WCHAR_WIDTH

  Indicate the size of the `wchar` type.

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_RENAME(GHOTIIO_CUTIL_NAME, _ ## NAME)
```

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

Notice, we cannot use GHOTIIO_CUTIL_CONCAT2(), because the preprocessor dies in some cases with nested use (see vector.template.c).

Parameters

NAME	The name which will be prepended with the `GHOTIIO_CUTIL_NAME`.

5.4.2.2 GHOTIIO_CUTIL_CONCAT2

```
#define GHOTIIO_CUTIL_CONCAT2(
              a,
              b ) GHOTIIO_CUTIL_CONCAT2_INNER(a,b)
```

Helper macro to concatenate the identifiers.

It requires two levels of processing.

This macro may be called directly.

Parameters

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

Returns

A call to the `GHOTIIO_CUTIL_CONCAT2_INNER()` macro.

5.4.2.3 GHOTIIO_CUTIL_CONCAT2_INNER

```
#define GHOTIIO_CUTIL_CONCAT2_INNER(
              a,
              b ) a ## b
```

Helper macro to concatenate the identifiers.

It reuires two levels of processing.

This macro should not be called directly. It should only be called by GHOTIIO_CUTIL_CONCAT2().

Parameters

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

Returns

The concatenation of `a` to `b`.

5.4.2.4 GHOTIIO_CUTIL_CONCAT3

```
#define GHOTIIO_CUTIL_CONCAT3(
              a,
              b,
              c ) GHOTIIO_CUTIL_CONCAT3_INNER(a,b,c)
```

Helper macro to concatenate the identifiers.

It requires two levels of processing.

This macro may be called directly.

Parameters

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

Returns

A call to the `GHOTIIO_CUTIL_CONCAT3_INNER()` macro.

5.4.2.5 GHOTIIO_CUTIL_CONCAT3_INNER

```
#define GHOTIIO_CUTIL_CONCAT3_INNER(
              a,
              b,
              c ) a ## b ## c
```

Helper macro to concatenate the identifiers.

It reuires two levels of processing.

This macro should not be called directly. It should only be called by GHOTIIO_CUTIL_CONCAT2().

Parameters

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

Returns

The concatenation of `a` to `b` to `c`..

5.4.2.6 GHOTIIO_CUTIL_NAME

```
#define GHOTIIO_CUTIL_NAME ghotiio_cutil_dev
```

Used in conjunction with the GHOTIIO_CUTIL...

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

5.4.2.7 GHOTIIO_CUTIL_RENAME

```
#define GHOTIIO_CUTIL_RENAME(
              a,
              b ) GHOTIIO_CUTIL_RENAME_INNER(a,b)
```

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

It requires two levels of processing.

Parameters

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

Returns

A call to the `GHOTIIO_CUTIL_RENAME_INNER()` macro.

5.4.2.8 GHOTIIO_CUTIL_RENAME_INNER

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

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

It requires two levels of processing.

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

Parameters

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

Returns

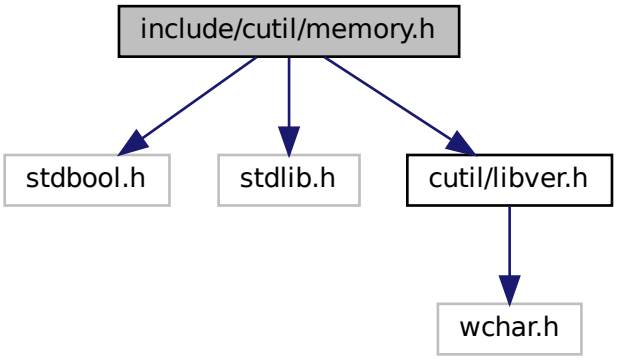
The concatenation of `a` to `b`.

5.5 include/cutil/memory.h File Reference

Header file for memory-related functions.

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

Include dependency graph for memory.h:



Functions

- void gcu_mem_start (void)

  Instruct Ghoti.io CUtils library that intercepted memory management calls should be logged to stderr.
- void gcu_mem_stop (void)

  Instruct Ghoti.io CUtils library that intercepted memory management calls should no longer be logged to stderr.
- void ∗ gcu_malloc_debug (size_t size, const char ∗file, size_t line)

  Cross-platform wrapper for the standard malloc() function.
- void ∗ gcu_calloc_debug (size_t nitems, size_t size, const char ∗file, size_t line)

  Cross-platform wrapper for the standard calloc() function.
- void ∗ gcu_realloc_debug (void ∗pointer, size_t size, const char ∗file, size_t line)

  Cross-platform wrapper for the standard realloc() function.
- void gcu_free_debug (void ∗pointer, const char ∗file, size_t line)

Wrapper for the standard free() function.

- void * gcu_malloc (size_t size)

  Cross-platform wrapper for the standard malloc() function.
- void * gcu_calloc (size_t nitems, size_t size)

  Cross-platform wrapper for the standard calloc() function.
- void * gcu_realloc (void *pointer, size_t size)

  Cross-platform wrapper for the standard realloc() function.
- void gcu_free (void *pointer)

  Wrapper for the standard free() function.

5.5.1 Detailed Description

Header file for memory-related functions.

For cross-platform memory functions, use the gcu_malloc(), gcu_calloc(), gcu_realloc(), and gcu_free() in this library.

To enable logging and debugging, define `GHOTIIO_CUTIL_ENABLE_MEMORY_DEBUG` before including this file. Then, all code compiled with this option will have memory logging enabled.

Logging to `stderr` is enabled by default when the afore-mention define is enabled. 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.

5.5.2 Function Documentation

5.5.2.1 gcu_calloc()

```
static void * gcu_calloc (
            size_t nitems,
            size_t size )  [inline]
```

Cross-platform wrapper for the standard calloc() function.

Parameters

nitems	The number of items to allocate.
size	The number of bytes in each item.

Returns

The beginning byte of the allocated memory.

5.5.2.2 gcu_calloc_debug()

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

Cross-platform wrapper for the standard calloc() function.

This function should not be called directly. Call gcu_calloc() instead.

Parameters

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

Returns

The beginning byte of the allocated memory.

5.5.2.3 gcu_free()

```
static void gcu_free (
            void * pointer )  [inline]
```

Wrapper for the standard free() function.

Parameters

pointer	The beginning byte of the currently allocated memory.

5.5.2.4 gcu_free_debug()

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

Wrapper for the standard free() function.

This function should not be called directly. Call gcu_free() instead.

Parameters

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

5.5.2.5 gcu_malloc()

```
static void * gcu_malloc (
            size_t size )  [inline]
```

Cross-platform wrapper for the standard malloc() function.

Parameters

size	The number of bytes requested.

Returns

The beginning byte of the allocated memory.

5.5.2.6 gcu_malloc_debug()

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

Cross-platform wrapper for the standard malloc() function.

This function should not be called directly. Call gcu_malloc() instead.

Parameters

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

Returns

The beginning byte of the allocated memory.

5.5.2.7 gcu_realloc()

```
static void * gcu_realloc (
            void * pointer,
            size_t size )  [inline]
```

Cross-platform wrapper for the standard realloc() function.

Parameters

pointer	The beginning byte of the currently allocated memory.
size	The newly requested size.

Returns

The beginning byte of the reallocated memory.

5.5.2.8 gcu_realloc_debug()

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

Cross-platform wrapper for the standard realloc() function.

This function should not be called directly. Call gcu_realloc() instead.

Parameters

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

Returns

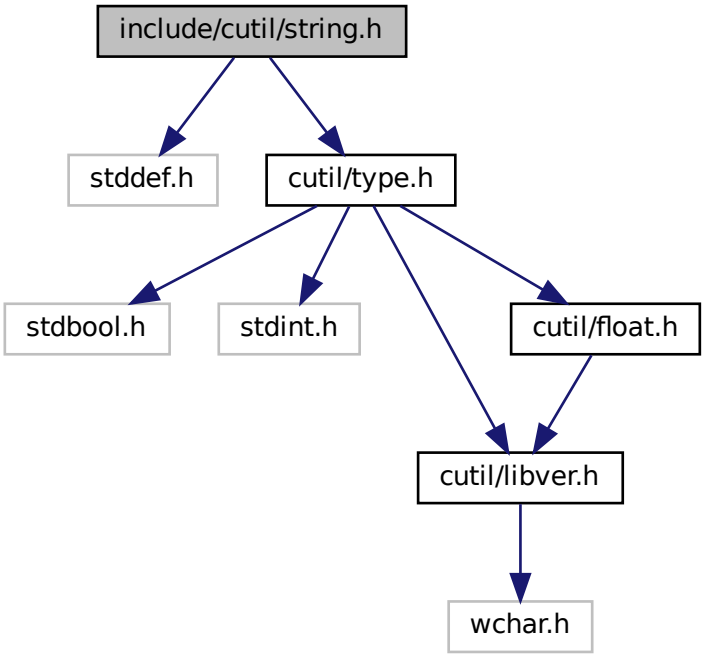
The beginning byte of the reallocated memory.

5.6 include/cutil/string.h File Reference

A collection of string-related functions.

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

Include dependency graph for string.h:



Functions

- uint32_t gcu_string_hash_32 (char const ∗str, size_t len)

  Helper function to wrap the hash function that produces a 32-bit number representing the hash.
- uint64_t gcu_string_hash_64 (char const ∗str, size_t len)

  Helper function to wrap the hash function that produces a 64-bit number representing the hash.
- void gcu_string_murmur3_32 (const void ∗key, size_t len, uint32_t seed, void ∗out)

  Get 32-bit hash using the MurmurHash3 by Appleby.
- void gcu_string_murmur3_x86_128 (const void ∗key, size_t len, uint32_t seed, void ∗out)

  Get 128-bit hash using the MurmurHash3 for x86 architecture by Appleby.
- void gcu_string_murmur3_x64_128 (const void ∗key, size_t len, uint32_t seed, void ∗out)

  Get 128-bit hash using the MurmurHash3 for x64 architecture by Appleby.

5.6.1 Detailed Description

A collection of string-related functions.

5.6.2 Function Documentation

5.6.2.1 gcu_string_hash_32()

```
uint32_t gcu_string_hash_32 (
            char const * str,
            size_t len )
```

Helper function to wrap the hash function that produces a 32-bit number representing the hash.

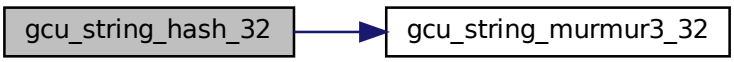
Parameters

str	A pointer to the string (or data block).
len	The length of the data in bytes.

Returns

A 32-bit number representing the value.

Here is the call graph for this function:



5.6.2.2 gcu_string_hash_64()

```
uint64_t gcu_string_hash_64 (
            char const * str,
            size_t len )
```

Helper function to wrap the hash function that produces a 64-bit number representing the hash.

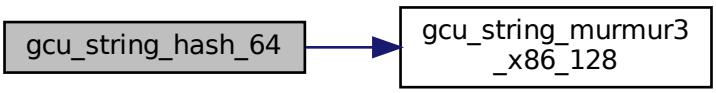
Parameters

str	A pointer to the string (or data block).
len	The length of the data in bytes.

Returns

A 64-bit number representing the value.

Here is the call graph for this function:



5.6.2.3 gcu_string_murmur3_32()

```
void gcu_string_murmur3_32 (
            const void * key,
            size_t len,
            uint32_t seed,
            void * out )
```

Get 32-bit hash using the MurmurHash3 by Appleby.

MurmurHash3 hashing algorithm, was created and put into the public domain by Austin Appleby, originally in C++.
https://github.com/aappleby/smhasher/blob/master/src/MurmurHash3.cpp

Parameters

key	A pointer to the start of the source data.
len	The size of the data in bytes.
seed	A seed value for the initial hash.
out	A pointer to a 32-bit (4-byte) buffer into which the hash may be written. The caller must supply the buffer.

5.6.2.4 gcu_string_murmur3_x64_128()

```
void gcu_string_murmur3_x64_128 (
            const void * key,
            size_t len,
            uint32_t seed,
            void * out )
```

Get 128-bit hash using the MurmurHash3 for x64 architecture by Appleby.

The x86 version does not produce the same hash as the x64 version, by design by Appleby.

MurmurHash3 hashing algorithm, was created and put into the public domain by Austin Appleby, originally in C++.
https://github.com/aappleby/smhasher/blob/master/src/MurmurHash3.cpp

Parameters

key	A pointer to the start of the source data.
len	The size of the data in bytes.
seed	A seed value for the initial hash.
out	A pointer to a 128-bit (16-byte) buffer into which the hash may be written. The caller must supply the buffer.

5.6.2.5 gcu_string_murmur3_x86_128()

```
void gcu_string_murmur3_x86_128 (
            const void * key,
            size_t len,
            uint32_t seed,
            void * out )
```

Get 128-bit hash using the MurmurHash3 for x86 architecture by Appleby.

The x86 version does not produce the same hash as the x64 version, by design by Appleby.

MurmurHash3 hashing algorithm, was created and put into the public domain by Austin Appleby, originally in C++.
https://github.com/aappleby/smhasher/blob/master/src/MurmurHash3.cpp

Parameters

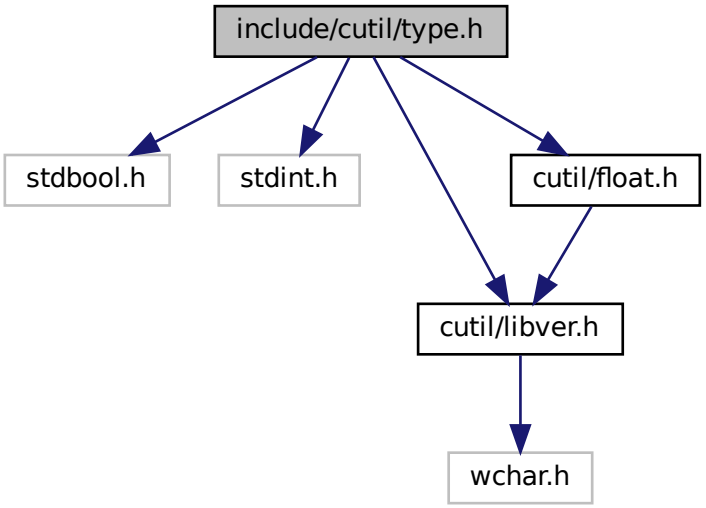
key	A pointer to the start of the source data.
len	The size of the data in bytes.
seed	A seed value for the initial hash.
out	A pointer to a 128-bit (16-byte) buffer into which the hash may be written. The caller must supply the buffer.

5.7 include/cutil/type.h File Reference

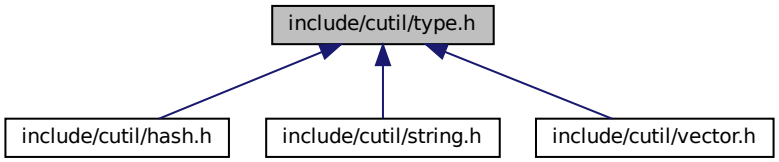
Type definitions and utilities for use by the Ghoti.io projects.

```
#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.

Macros

- #define GCU_TYPE64_P(val) ((GCU_Type64_Union) {.p = val})

  Create a 64-bit union variable with the type `void *`.
- #define GCU_TYPE64_UI64(val) ((GCU_Type64_Union) {.ui64 = val})

  Create a 64-bit union variable with the type `uint64_t`.
- #define GCU_TYPE64_UI32(val) ((GCU_Type64_Union) {.ui32 = val})

  Create a 64-bit union variable with the type `uint32_t`.
- #define GCU_TYPE64_UI16(val) ((GCU_Type64_Union) {.ui16 = val})

  Create a 64-bit union variable with the type `uint16_t`.
- #define GCU_TYPE64_UI8(val) ((GCU_Type64_Union) {.ui8 = val})

  Create a 64-bit union variable with the type `uint8_t`.
- #define GCU_TYPE64_I64(val) ((GCU_Type64_Union) {.i64 = val})

  Create a 64-bit union variable with the type `int64_t`.
- #define GCU_TYPE64_I32(val) ((GCU_Type64_Union) {.i32 = val})

  Create a 64-bit union variable with the type `int32_t`.
- #define GCU_TYPE64_I16(val) ((GCU_Type64_Union) {.i16 = val})

  Create a 64-bit union variable with the type `int16_t`.
- #define GCU_TYPE64_I8(val) ((GCU_Type64_Union) {.i8 = val})

  Create a 64-bit union variable with the type `int8_t`.
- #define GCU_TYPE64_F64(val) ((GCU_Type64_Union) {.f64 = val})

  Create a 64-bit union variable with the type float with 64 bits.
- #define GCU_TYPE64_F32(val) ((GCU_Type64_Union) {.f32 = val})

  Create a 64-bit union variable with the type float with 32 bits.
- #define GCU_TYPE64_WC(val) ((GCU_Type64_Union) {.wc = val})

  Create a 64-bit union variable with the type `wchar_t`.
- #define GCU_TYPE64_C(val) ((GCU_Type64_Union) {.c = val})

  Create a 64-bit union variable with the type `char`.
- #define GCU_TYPE32_UI32(val) ((GCU_Type32_Union) {.ui32 = val})

  Create a 32-bit union variable with the type `uint32_t`.
- #define GCU_TYPE32_UI16(val) ((GCU_Type32_Union) {.ui16 = val})

  Create a 32-bit union variable with the type `uint16_t`.
- #define GCU_TYPE32_UI8(val) ((GCU_Type32_Union) {.ui8 = val})

  Create a 32-bit union variable with the type `uint8_t`.
- #define GCU_TYPE32_I32(val) ((GCU_Type32_Union) {.i32 = val})

  Create a 32-bit union variable with the type `int32_t`.
- #define GCU_TYPE32_I16(val) ((GCU_Type32_Union) {.i16 = val})

  Create a 32-bit union variable with the type `int16_t`.
- #define GCU_TYPE32_I8(val) ((GCU_Type32_Union) {.i8 = val})

  Create a 32-bit union variable with the type `int8_t`.
- #define GCU_TYPE32_F32(val) ((GCU_Type32_Union) {.f32 = val})

  Create a 32-bit union variable with the type float with 32 bits.
- #define GCU_TYPE32_WC(val) ((GCU_Type32_Union) {.wc = val})

  Create a 32-bit union variable with the type `wchar_t`.
- #define GCU_TYPE32_C(val) ((GCU_Type32_Union) {.c = val})

  Create a 32-bit union variable with the type `char`.
- #define GCU_TYPE16_UI16(val) ((GCU_Type16_Union) {.ui16 = val})

  Create a 16-bit union variable with the type `uint16_t`.
- #define GCU_TYPE16_UI8(val) ((GCU_Type16_Union) {.ui8 = val})

  Create a 16-bit union variable with the type `uint8_t`.
- #define GCU_TYPE16_I16(val) ((GCU_Type16_Union) {.i16 = val})

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

- #define GCU_TYPE16_I8(val) ((GCU_Type16_Union) {.i8 = val})

  Create a 16-bit union variable with the type `int8_t`.
- #define GCU_TYPE16_C(val) ((GCU_Type16_Union) {.c = val})

  Create a 16-bit union variable with the type `char`.
- #define GCU_TYPE8_UI8(val) ((GCU_Type8_Union) {.ui8 = val})

  Create a 8-bit union variable with the type `uint8_t`.
- #define GCU_TYPE8_I8(val) ((GCU_Type8_Union) {.i8 = val})

  Create a 8-bit union variable with the type `int8_t`.
- #define GCU_TYPE8_C(val) ((GCU_Type8_Union) {.c = val})

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

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_wc (wchar_t val)

  Create a 64-bit union variable with the type `wchar_t`.
- 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_wc (wchar_t val)

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

5.7.1 Detailed Description

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

5.7.2 Macro Definition Documentation

5.7.2.1 GCU_TYPE16_C

```
#define GCU_TYPE16_C(
              val ) ((GCU_Type16_Union) {.c = val})
```

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

This `#define` is a compound literal. It is allowed in C but not C++. There is a corresponding function for use in C++.

See also

gcu_type16_c()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.2.2 GCU_TYPE16_I16

```
#define GCU_TYPE16_I16(
              val ) ((GCU_Type16_Union) {.i16 = val})
```

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

This `#define` is a compound literal. It is allowed in C but not C++. There is a corresponding function for use in C++.

See also

gcu_type16_i16()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.2.3 GCU_TYPE16_I8

```
#define GCU_TYPE16_I8(
              val ) ((GCU_Type16_Union) {.i8 = val})
```

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

This `#define` is a compound literal. It is allowed in C but not C++. There is a corresponding function for use in C++.

See also

gcu_type16_i8()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.2.4 GCU_TYPE16_UI16

```
#define GCU_TYPE16_UI16(
              val ) ((GCU_Type16_Union) {.ui16 = val})
```

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

This `#define` is a compound literal. It is allowed in C but not C++. There is a corresponding function for use in C++.

See also

gcu_type16_ui16()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.2.5 GCU_TYPE16_UI8

```
#define GCU_TYPE16_UI8(
              val ) ((GCU_Type16_Union) {.ui8 = val})
```

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

This `#define` is a compound literal. It is allowed in C but not C++. There is a corresponding function for use in C++.

See also

gcu_type16_ui8()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.2.6 GCU_TYPE32_C

```
#define GCU_TYPE32_C(
              val ) ((GCU_Type32_Union) {.c = val})
```

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

This `#define` is a compound literal. It is allowed in C but not C++. There is a corresponding function for use in C++.

See also

gcu_type32_c()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.2.7 GCU_TYPE32_F32

```
#define GCU_TYPE32_F32(
              val ) ((GCU_Type32_Union) {.f32 = val})
```

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

This `#define` is a compound literal. It is allowed in C but not C++. There is a corresponding function for use in C++.

See also

gcu_type32_f32()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.2.8 GCU_TYPE32_I16

```
#define GCU_TYPE32_I16(
              val ) ((GCU_Type32_Union) {.i16 = val})
```

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

This `#define` is a compound literal. It is allowed in C but not C++. There is a corresponding function for use in C++.

See also

gcu_type32_i16()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.2.9 GCU_TYPE32_I32

```
#define GCU_TYPE32_I32(
              val ) ((GCU_Type32_Union) {.i32 = val})
```

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

This `#define` is a compound literal. It is allowed in C but not C++. There is a corresponding function for use in C++.

See also

gcu_type32_i32()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.2.10 GCU_TYPE32_I8

```
#define GCU_TYPE32_I8(
              val ) ((GCU_Type32_Union) {.i8 = val})
```

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

This `#define` is a compound literal. It is allowed in C but not C++. There is a corresponding function for use in C++.

See also

gcu_type32_i8()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.2.11 GCU_TYPE32_UI16

```
#define GCU_TYPE32_UI16(
              val ) ((GCU_Type32_Union) {.ui16 = val})
```

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

This `#define` is a compound literal. It is allowed in C but not C++. There is a corresponding function for use in C++.

See also

gcu_type32_ui16()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.2.12 GCU_TYPE32_UI32

```
#define GCU_TYPE32_UI32(
              val ) ((GCU_Type32_Union) {.ui32 = val})
```

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

This `#define` is a compound literal. It is allowed in C but not C++. There is a corresponding function for use in C++.

See also

gcu_type32_ui32()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.2.13 GCU_TYPE32_UI8

```
#define GCU_TYPE32_UI8(
              val ) ((GCU_Type32_Union) {.ui8 = val})
```

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

This `#define` is a compound literal. It is allowed in C but not C++. There is a corresponding function for use in C++.

See also

gcu_type32_ui8()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.2.14 GCU_TYPE32_WC

```
#define GCU_TYPE32_WC(
              val ) ((GCU_Type32_Union) {.wc = val})
```

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

This `#define` is a compound literal. It is allowed in C but not C++. There is a corresponding function for use in C++.

See also

gcu_type32_wc()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.2.15 GCU_TYPE64_C

```
#define GCU_TYPE64_C(
              val ) ((GCU_Type64_Union) {.c = val})
```

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

This `#define` is a compound literal. It is allowed in C but not C++. There is a corresponding function for use in C++.

See also

gcu_type64_c()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.2.16 GCU_TYPE64_F32

```
#define GCU_TYPE64_F32(
              val ) ((GCU_Type64_Union) {.f32 = val})
```

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

This `#define` is a compound literal. It is allowed in C but not C++. There is a corresponding function for use in C++.

See also

gcu_type64_f32()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.2.17 GCU_TYPE64_F64

```
#define GCU_TYPE64_F64(
              val ) ((GCU_Type64_Union) {.f64 = val})
```

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

This `#define` is a compound literal. It is allowed in C but not C++. There is a corresponding function for use in C++.

See also

gcu_type64_f64()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.2.18 GCU_TYPE64_I16

```
#define GCU_TYPE64_I16(
              val ) ((GCU_Type64_Union) {.i16 = val})
```

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

This `#define` is a compound literal. It is allowed in C but not C++. There is a corresponding function for use in C++.

See also

gcu_type64_i16()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.2.19 GCU_TYPE64_I32

```
#define GCU_TYPE64_I32(
              val ) ((GCU_Type64_Union) {.i32 = val})
```

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

This `#define` is a compound literal. It is allowed in C but not C++. There is a corresponding function for use in C++.

See also

gcu_type64_i32()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.2.20 GCU_TYPE64_I64

```
#define GCU_TYPE64_I64(
              val ) ((GCU_Type64_Union) {.i64 = val})
```

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

This `#define` is a compound literal. It is allowed in C but not C++. There is a corresponding function for use in C++.

See also

gcu_type64_i64()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.2.21 GCU_TYPE64_I8

```
#define GCU_TYPE64_I8(
              val ) ((GCU_Type64_Union) {.i8 = val})
```

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

This `#define` is a compound literal. It is allowed in C but not C++. There is a corresponding function for use in C++.

See also

gcu_type64_i8()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.2.22 GCU_TYPE64_P

```
#define GCU_TYPE64_P(
              val ) ((GCU_Type64_Union) {.p = val})
```

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

This `#define` is a compound literal. It is allowed in C but not C++. There is a corresponding function for use in C++.

See also

gcu_type64_p()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.2.23 GCU_TYPE64_UI16

```
#define GCU_TYPE64_UI16(
              val ) ((GCU_Type64_Union) {.ui16 = val})
```

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

This `#define` is a compound literal. It is allowed in C but not C++. There is a corresponding function for use in C++.

See also

gcu_type64_ui16()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.2.24 GCU_TYPE64_UI32

```
#define GCU_TYPE64_UI32(
              val ) ((GCU_Type64_Union) {.ui32 = val})
```

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

This `#define` is a compound literal. It is allowed in C but not C++. There is a corresponding function for use in C++.

See also

gcu_type64_ui32()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.2.25 GCU_TYPE64_UI64

```
#define GCU_TYPE64_UI64(
              val ) ((GCU_Type64_Union) {.ui64 = val})
```

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

This `#define` is a compound literal. It is allowed in C but not C++. There is a corresponding function for use in C++.

See also

gcu_type64_ui64()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.2.26 GCU_TYPE64_UI8

```
#define GCU_TYPE64_UI8(
              val ) ((GCU_Type64_Union) {.ui8 = val})
```

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

This `#define` is a compound literal. It is allowed in C but not C++. There is a corresponding function for use in C++.

See also

gcu_type64_ui8()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.2.27 GCU_TYPE64_WC

```
#define GCU_TYPE64_WC(
              val ) ((GCU_Type64_Union) {.wc = val})
```

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

This `#define` is a compound literal. It is allowed in C but not C++. There is a corresponding function for use in C++.

See also

gcu_type64_wc()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.2.28 GCU_TYPE8_C

```
#define GCU_TYPE8_C(
              val ) ((GCU_Type8_Union) {.c = val})
```

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

This `#define` is a compound literal. It is allowed in C but not C++. There is a corresponding function for use in C++.

See also

gcu_type8_c()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.2.29 GCU_TYPE8_I8

```
#define GCU_TYPE8_I8(
              val ) ((GCU_Type8_Union) {.i8 = val})
```

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

This `#define` is a compound literal. It is allowed in C but not C++. There is a corresponding function for use in C++.

See also

gcu_type8_i8()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.2.30 GCU_TYPE8_UI8

```
#define GCU_TYPE8_UI8(
              val ) ((GCU_Type8_Union) {.ui8 = val})
```

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

This `#define` is a compound literal. It is allowed in C but not C++. There is a corresponding function for use in C++.

See also

gcu_type8_ui8()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.3 Function Documentation

5.7.3.1 gcu_type16_c()

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

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

This function is provided as a helper in C++ because C++ does not allow the use of compound literals. If in C, use the `#define`.

See also

GCU_TYPE16_C()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.3.2 gcu_type16_i16()

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

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

This function is provided as a helper in C++ because C++ does not allow the use of compound literals. If in C, use the `#define`.

See also

GCU_TYPE16_I16()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.3.3 gcu_type16_i8()

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

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

This function is provided as a helper in C++ because C++ does not allow the use of compound literals. If in C, use the `#define`.

See also

GCU_TYPE16_I8()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.3.4 gcu_type16_ui16()

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

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

This function is provided as a helper in C++ because C++ does not allow the use of compound literals. If in C, use the `#define`.

See also

GCU_TYPE16_UI16()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.3.5 gcu_type16_ui8()

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

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

This function is provided as a helper in C++ because C++ does not allow the use of compound literals. If in C, use the `#define`.

See also

GCU_TYPE16_UI8()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.3.6 gcu_type32_c()

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

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

This function is provided as a helper in C++ because C++ does not allow the use of compound literals. If in C, use the `#define`.

See also

GCU_TYPE32_C()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.3.7 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.

This function is provided as a helper in C++ because C++ does not allow the use of compound literals. If in C, use the `#define`.

See also

GCU_TYPE32_F32()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.3.8 gcu_type32_i16()

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

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

This function is provided as a helper in C++ because C++ does not allow the use of compound literals. If in C, use the `#define`.

See also

GCU_TYPE32_I16()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.3.9 gcu_type32_i32()

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

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

This function is provided as a helper in C++ because C++ does not allow the use of compound literals. If in C, use the `#define`.

See also

GCU_TYPE32_I32()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.3.10 gcu_type32_i8()

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

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

This function is provided as a helper in C++ because C++ does not allow the use of compound literals. If in C, use the `#define`.

See also

GCU_TYPE32_I8()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.3.11 gcu_type32_ui16()

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

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

This function is provided as a helper in C++ because C++ does not allow the use of compound literals. If in C, use the `#define`.

See also

GCU_TYPE32_UI16()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.3.12 gcu_type32_ui32()

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

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

This function is provided as a helper in C++ because C++ does not allow the use of compound literals. If in C, use the `#define`.

See also

GCU_TYPE32_UI32()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.3.13 gcu_type32_ui8()

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

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

This function is provided as a helper in C++ because C++ does not allow the use of compound literals. If in C, use the `#define`.

See also

GCU_TYPE32_UI8()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.3.14 gcu_type32_wc()

```
GCU_Type32_Union gcu_type32_wc (
            wchar_t val )
```

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

This function is provided as a helper in C++ because C++ does not allow the use of compound literals. If in C, use the `#define`.

See also

GCU_TYPE32_WC()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.3.15 gcu_type64_c()

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

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

This function is provided as a helper in C++ because C++ does not allow the use of compound literals. If in C, use the `#define`.

See also

GCU_TYPE64_C()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.3.16 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.

This function is provided as a helper in C++ because C++ does not allow the use of compound literals. If in C, use the `#define`.

See also

GCU_TYPE64_F32()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.3.17 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.

This function is provided as a helper in C++ because C++ does not allow the use of compound literals. If in C, use the `#define`.

See also

GCU_TYPE64_F64()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.3.18 gcu_type64_i16()

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

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

This function is provided as a helper in C++ because C++ does not allow the use of compound literals. If in C, use the `#define`.

See also

GCU_TYPE64_I16()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.3.19 gcu_type64_i32()

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

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

This function is provided as a helper in C++ because C++ does not allow the use of compound literals. If in C, use the `#define`.

See also

GCU_TYPE64_I32()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.3.20 gcu_type64_i64()

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

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

This function is provided as a helper in C++ because C++ does not allow the use of compound literals. If in C, use the `#define`.

See also

GCU_TYPE64_I64()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.3.21 gcu_type64_i8()

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

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

This function is provided as a helper in C++ because C++ does not allow the use of compound literals. If in C, use the `#define`.

See also

GCU_TYPE64_I8()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.3.22 gcu_type64_p()

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

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

This function is provided as a helper in C++ because C++ does not allow the use of compound literals. If in C, use the `#define`.

See also

GCU_TYPE64_P()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.3.23 gcu_type64_ui16()

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

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

This function is provided as a helper in C++ because C++ does not allow the use of compound literals. If in C, use the `#define`.

See also

GCU_TYPE64_UI16()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.3.24 gcu_type64_ui32()

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

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

This function is provided as a helper in C++ because C++ does not allow the use of compound literals. If in C, use the `#define`.

See also

GCU_TYPE64_UI32()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.3.25 gcu_type64_ui64()

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

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

This function is provided as a helper in C++ because C++ does not allow the use of compound literals. If in C, use the `#define`.

See also

GCU_TYPE64_UI64()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.3.26 gcu_type64_ui8()

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

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

This function is provided as a helper in C++ because C++ does not allow the use of compound literals. If in C, use the `#define`.

See also

GCU_TYPE64_UI8()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.3.27 gcu_type64_wc()

```
GCU_Type64_Union gcu_type64_wc (
            wchar_t val )
```

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

This function is provided as a helper in C++ because C++ does not allow the use of compound literals. If in C, use the `#define`.

See also

GCU_TYPE64_WC()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.3.28 gcu_type8_c()

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

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

This function is provided as a helper in C++ because C++ does not allow the use of compound literals. If in C, use the `#define`.

See also

GCU_TYPE8_C()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.3.29 gcu_type8_i8()

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

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

This function is provided as a helper in C++ because C++ does not allow the use of compound literals. If in C, use the `#define`.

See also

GCU_TYPE8_I8()

Parameters

val	The value to put into the union.

Returns

The union variable.

5.7.3.30 gcu_type8_ui8()

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

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

This function is provided as a helper in C++ because C++ does not allow the use of compound literals. If in C, use the `#define`.

See also

GCU_TYPE8_UI8()

Parameters

val	The value to put into the union.

Returns

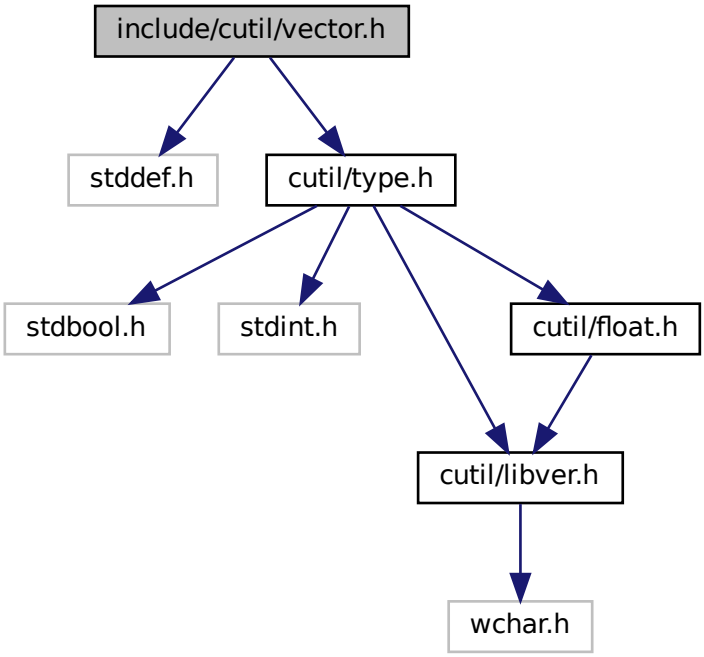
The union variable.

5.8 include/cutil/vector.h File Reference

A simple vector implementation.

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

Include dependency graph for vector.h:



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.

5.8.1 Detailed Description

A simple vector implementation.

5.8.2 Function Documentation

5.8.2.1 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

vector	The vector structure on which to operate.
value	The item to append to the end of the vector.

Returns

`true` on success, `false` otherwise.

5.8.2.2 gcu_vector16_count()

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

Get a count of entries in the vector.

Parameters

vector	The vector structure on which to operate.

Returns

The count of entries in the vector.

5.8.2.3 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

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

Returns

A struct containing the vector information.

5.8.2.4 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

vector	The vector structure to be destroyed.

5.8.2.5 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

vector	The vector structure on which to operate.
value	The item to append to the end of the vector.

Returns

`true` on success, `false` otherwise.

5.8.2.6 gcu_vector32_count()

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

Get a count of entries in the vector.

Parameters

vector	The vector structure on which to operate.

Returns

The count of entries in the vector.

5.8.2.7 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

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

Returns

A struct containing the vector information.

5.8.2.8 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

vector	The vector structure to be destroyed.

5.8.2.9 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

vector	The vector structure on which to operate.
value	The item to append to the end of the vector.

Returns

`true` on success, `false` otherwise.

5.8.2.10 gcu_vector64_count()

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

Get a count of entries in the vector.

Parameters

vector	The vector structure on which to operate.

Returns

The count of entries in the vector.

5.8.2.11 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

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

Returns

A struct containing the vector information.

5.8.2.12 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

vector	The vector structure to be destroyed.

5.8.2.13 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

vector	The vector structure on which to operate.
value	The item to append to the end of the vector.

Returns

`true` on success, `false` otherwise.

5.8.2.14 gcu_vector8_count()

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

Get a count of entries in the vector.

Parameters

vector	The vector structure on which to operate.

Returns

The count of entries in the vector.

5.8.2.15 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

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

Returns

A struct containing the vector information.

5.8.2.16 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

vector	The vector structure to be destroyed.

Index