

Ghoti.io CUtil

0.1

Generated by Doxygen 1.9.1



|   |          |
|---|----------|
| <b>1 Ghoti.io CUtil Library</b>           | <b>1</b> |
| 1.0.1 Overview                            | 1        |
| 1.0.2 Installation                        | 1        |
| 1.0.2.1 Build From Source                 | 1        |
| 1.0.3 Compiling With The Library          | 1        |
| <b>2 Class Index</b>                      | <b>3</b> |
| 2.1 Class List                            | 3        |
| <b>3 File Index</b>                       | <b>5</b> |
| 3.1 File List                             | 5        |
| <b>4 Class Documentation</b>              | <b>7</b> |
| 4.1 GCU_Hash16_Cell Struct Reference      | 7        |
| 4.1.1 Detailed Description                | 8        |
| 4.2 GCU_Hash16_Iterator Struct Reference  | 8        |
| 4.2.1 Detailed Description                | 9        |
| 4.3 GCU_Hash16_Table Struct Reference     | 9        |
| 4.3.1 Detailed Description                | 10       |
| 4.4 GCU_Hash16_Value Struct Reference     | 10       |
| 4.4.1 Detailed Description                | 11       |
| 4.5 GCU_Hash32_Cell Struct Reference      | 11       |
| 4.5.1 Detailed Description                | 12       |
| 4.6 GCU_Hash32_Iterator Struct Reference  | 12       |
| 4.6.1 Detailed Description                | 13       |
| 4.7 GCU_Hash32_Table Struct Reference     | 13       |
| 4.7.1 Detailed Description                | 14       |
| 4.8 GCU_Hash32_Value Struct Reference     | 14       |
| 4.8.1 Detailed Description                | 15       |
| 4.9 GCU_Hash64_Cell Struct Reference      | 15       |
| 4.9.1 Detailed Description                | 16       |
| 4.10 GCU_Hash64_Iterator Struct Reference | 16       |
| 4.10.1 Detailed Description               | 17       |
| 4.11 GCU_Hash64_Table Struct Reference    | 17       |
| 4.11.1 Detailed Description               | 18       |
| 4.12 GCU_Hash64_Value Struct Reference    | 18       |
| 4.12.1 Detailed Description               | 19       |
| 4.13 GCU_Hash8_Cell Struct Reference      | 19       |
| 4.13.1 Detailed Description               | 20       |
| 4.14 GCU_Hash8_Iterator Struct Reference  | 20       |
| 4.14.1 Detailed Description               | 21       |
| 4.15 GCU_Hash8_Table Struct Reference     | 21       |
| 4.15.1 Detailed Description               | 22       |

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

|   |    |
|---|----|
| 5.3.2.11 gcu_hash32_count()               | 40 |
| 5.3.2.12 gcu_hash32_create()              | 40 |
| 5.3.2.13 gcu_hash32_destroy()             | 41 |
| 5.3.2.14 gcu_hash32_get()                 | 41 |
| 5.3.2.15 gcu_hash32_iterator_get()        | 41 |
| 5.3.2.16 gcu_hash32_iterator_next()       | 43 |
| 5.3.2.17 gcu_hash32_remove()              | 43 |
| 5.3.2.18 gcu_hash32_set()                 | 44 |
| 5.3.2.19 gcu_hash64_contains()            | 44 |
| 5.3.2.20 gcu_hash64_count()               | 45 |
| 5.3.2.21 gcu_hash64_create()              | 45 |
| 5.3.2.22 gcu_hash64_destroy()             | 45 |
| 5.3.2.23 gcu_hash64_get()                 | 46 |
| 5.3.2.24 gcu_hash64_iterator_get()        | 46 |
| 5.3.2.25 gcu_hash64_iterator_next()       | 46 |
| 5.3.2.26 gcu_hash64_remove()              | 47 |
| 5.3.2.27 gcu_hash64_set()                 | 47 |
| 5.3.2.28 gcu_hash8_contains()             | 48 |
| 5.3.2.29 gcu_hash8_count()                | 48 |
| 5.3.2.30 gcu_hash8_create()               | 49 |
| 5.3.2.31 gcu_hash8_destroy()              | 49 |
| 5.3.2.32 gcu_hash8_get()                  | 49 |
| 5.3.2.33 gcu_hash8_iterator_get()         | 50 |
| 5.3.2.34 gcu_hash8_iterator_next()        | 50 |
| 5.3.2.35 gcu_hash8_remove()               | 51 |
| 5.3.2.36 gcu_hash8_set()                  | 51 |
| 5.4 include/cutil/libver.h File Reference | 51 |
| 5.4.1 Detailed Description                | 52 |
| 5.4.2 Macro Definition Documentation      | 52 |
| 5.4.2.1 GHOTIIO_CUTIL                     | 52 |
| 5.4.2.2 GHOTIIO_CUTIL_CONCAT2             | 53 |
| 5.4.2.3 GHOTIIO_CUTIL_CONCAT2_INNER       | 53 |
| 5.4.2.4 GHOTIIO_CUTIL_CONCAT3             | 54 |
| 5.4.2.5 GHOTIIO_CUTIL_CONCAT3_INNER       | 54 |
| 5.4.2.6 GHOTIIO_CUTIL_NAME                | 55 |
| 5.4.2.7 GHOTIIO_CUTIL_RENAME              | 55 |
| 5.4.2.8 GHOTIIO_CUTIL_RENAME_INNER        | 55 |
| 5.5 include/cutil/string.h File Reference | 56 |
| 5.5.1 Detailed Description                | 57 |
| 5.5.2 Function Documentation              | 57 |
| 5.5.2.1 gcu_string_hash_32()              | 57 |
| 5.5.2.2 gcu_string_hash_64()              | 58 |

|   |    |
|---|----|
| 5.5.2.3 gcu_string_murmur3_32()         | 58 |
| 5.5.2.4 gcu_string_murmur3_x64_128()    | 59 |
| 5.5.2.5 gcu_string_murmur3_x86_128()    | 59 |
| 5.6 include/cutil/type.h File Reference | 60 |
| 5.6.1 Detailed Description              | 63 |
| 5.6.2 Macro Definition Documentation    | 63 |
| 5.6.2.1 GCU_TYPE16_C                    | 63 |
| 5.6.2.2 GCU_TYPE16_I16                  | 64 |
| 5.6.2.3 GCU_TYPE16_I8                   | 64 |
| 5.6.2.4 GCU_TYPE16_UI16                 | 64 |
| 5.6.2.5 GCU_TYPE16_UI8                  | 65 |
| 5.6.2.6 GCU_TYPE32_C                    | 65 |
| 5.6.2.7 GCU_TYPE32_F32                  | 66 |
| 5.6.2.8 GCU_TYPE32_I16                  | 66 |
| 5.6.2.9 GCU_TYPE32_I32                  | 67 |
| 5.6.2.10 GCU_TYPE32_I8                  | 67 |
| 5.6.2.11 GCU_TYPE32_UI16                | 68 |
| 5.6.2.12 GCU_TYPE32_UI32                | 68 |
| 5.6.2.13 GCU_TYPE32_UI8                 | 69 |
| 5.6.2.14 GCU_TYPE64_C                   | 69 |
| 5.6.2.15 GCU_TYPE64_F32                 | 70 |
| 5.6.2.16 GCU_TYPE64_F64                 | 70 |
| 5.6.2.17 GCU_TYPE64_I16                 | 71 |
| 5.6.2.18 GCU_TYPE64_I32                 | 71 |
| 5.6.2.19 GCU_TYPE64_I64                 | 72 |
| 5.6.2.20 GCU_TYPE64_I8                  | 72 |
| 5.6.2.21 GCU_TYPE64_P                   | 73 |
| 5.6.2.22 GCU_TYPE64_UI16                | 73 |
| 5.6.2.23 GCU_TYPE64_UI32                | 74 |
| 5.6.2.24 GCU_TYPE64_UI64                | 74 |
| 5.6.2.25 GCU_TYPE64_UI8                 | 75 |
| 5.6.2.26 GCU_TYPE8_C                    | 75 |
| 5.6.2.27 GCU_TYPE8_I8                   | 76 |
| 5.6.2.28 GCU_TYPE8_UI8                  | 76 |
| 5.6.3 Function Documentation            | 77 |
| 5.6.3.1 gcu_type16_c()                  | 77 |
| 5.6.3.2 gcu_type16_i16()                | 77 |
| 5.6.3.3 gcu_type16_i8()                 | 78 |
| 5.6.3.4 gcu_type16_ui16()               | 78 |
| 5.6.3.5 gcu_type16_ui8()                | 79 |
| 5.6.3.6 gcu_type32_c()                  | 79 |
| 5.6.3.7 gcu_type32_f32()                | 80 |

|   |     |
|---|-----|
| 5.6.3.8 gcu_type32_i16()                  | 80  |
| 5.6.3.9 gcu_type32_i32()                  | 81  |
| 5.6.3.10 gcu_type32_i8()                  | 81  |
| 5.6.3.11 gcu_type32_ui16()                | 82  |
| 5.6.3.12 gcu_type32_ui32()                | 82  |
| 5.6.3.13 gcu_type32_ui8()                 | 83  |
| 5.6.3.14 gcu_type64_c()                   | 83  |
| 5.6.3.15 gcu_type64_f32()                 | 84  |
| 5.6.3.16 gcu_type64_f64()                 | 84  |
| 5.6.3.17 gcu_type64_i16()                 | 85  |
| 5.6.3.18 gcu_type64_i32()                 | 85  |
| 5.6.3.19 gcu_type64_i64()                 | 86  |
| 5.6.3.20 gcu_type64_i8()                  | 86  |
| 5.6.3.21 gcu_type64_p()                   | 87  |
| 5.6.3.22 gcu_type64_ui16()                | 87  |
| 5.6.3.23 gcu_type64_ui32()                | 88  |
| 5.6.3.24 gcu_type64_ui64()                | 88  |
| 5.6.3.25 gcu_type64_ui8()                 | 89  |
| 5.6.3.26 gcu_type8_c()                    | 89  |
| 5.6.3.27 gcu_type8_i8()                   | 90  |
| 5.6.3.28 gcu_type8_ui8()                  | 90  |
| 5.7 include/cutil/vector.h File Reference | 91  |
| 5.7.1 Detailed Description                | 92  |
| 5.7.2 Function Documentation              | 93  |
| 5.7.2.1 gcu_vector16_append()             | 93  |
| 5.7.2.2 gcu_vector16_count()              | 93  |
| 5.7.2.3 gcu_vector16_create()             | 93  |
| 5.7.2.4 gcu_vector16_destroy()            | 94  |
| 5.7.2.5 gcu_vector32_append()             | 94  |
| 5.7.2.6 gcu_vector32_count()              | 95  |
| 5.7.2.7 gcu_vector32_create()             | 95  |
| 5.7.2.8 gcu_vector32_destroy()            | 96  |
| 5.7.2.9 gcu_vector64_append()             | 96  |
| 5.7.2.10 gcu_vector64_count()             | 96  |
| 5.7.2.11 gcu_vector64_create()            | 97  |
| 5.7.2.12 gcu_vector64_destroy()           | 97  |
| 5.7.2.13 gcu_vector8_append()             | 97  |
| 5.7.2.14 gcu_vector8_count()              | 98  |
| 5.7.2.15 gcu_vector8_create()             | 98  |
| 5.7.2.16 gcu_vector8_destroy()            | 99  |
| 5.8 src/debug.c File Reference            | 99  |
| 5.8.1 Function Documentation              | 100 |

|   |     |
|---|-----|
| 5.8.1.1 gcu_malloc()                      | 100 |
| 5.8.1.2 gcu_free()                        | 101 |
| 5.8.1.3 gcu_malloc()                      | 101 |
| 5.8.1.4 gcu_realloc()                     | 101 |
| 5.9 src/float_identifier.c File Reference | 102 |
| 5.9.1 Detailed Description                | 102 |
| 5.10 src/hash.c File Reference            | 103 |
| 5.11 src/string.c File Reference          | 103 |
| 5.11.1 Function Documentation             | 104 |
| 5.11.1.1 gcu_string_hash_32()             | 105 |
| 5.11.1.2 gcu_string_hash_64()             | 105 |
| 5.11.1.3 gcu_string_murmur3_32()          | 106 |
| 5.11.1.4 gcu_string_murmur3_x64_128()     | 106 |
| 5.11.1.5 gcu_string_murmur3_x86_128()     | 108 |
| 5.12 src/type.c File Reference            | 108 |
| 5.12.1 Function Documentation             | 110 |
| 5.12.1.1 gcu_type16_c()                   | 110 |
| 5.12.1.2 gcu_type16_i16()                 | 111 |
| 5.12.1.3 gcu_type16_i8()                  | 111 |
| 5.12.1.4 gcu_type16_ui16()                | 112 |
| 5.12.1.5 gcu_type16_ui8()                 | 112 |
| 5.12.1.6 gcu_type32_c()                   | 113 |
| 5.12.1.7 gcu_type32_f32()                 | 113 |
| 5.12.1.8 gcu_type32_i16()                 | 114 |
| 5.12.1.9 gcu_type32_i32()                 | 114 |
| 5.12.1.10 gcu_type32_i8()                 | 115 |
| 5.12.1.11 gcu_type32_ui16()               | 115 |
| 5.12.1.12 gcu_type32_ui32()               | 116 |
| 5.12.1.13 gcu_type32_ui8()                | 116 |
| 5.12.1.14 gcu_type64_c()                  | 117 |
| 5.12.1.15 gcu_type64_f32()                | 117 |
| 5.12.1.16 gcu_type64_f64()                | 118 |
| 5.12.1.17 gcu_type64_i16()                | 118 |
| 5.12.1.18 gcu_type64_i32()                | 119 |
| 5.12.1.19 gcu_type64_i64()                | 119 |
| 5.12.1.20 gcu_type64_i8()                 | 120 |
| 5.12.1.21 gcu_type64_p()                  | 120 |
| 5.12.1.22 gcu_type64_ui16()               | 121 |
| 5.12.1.23 gcu_type64_ui32()               | 121 |
| 5.12.1.24 gcu_type64_ui64()               | 122 |
| 5.12.1.25 gcu_type64_ui8()                | 122 |
| 5.12.1.26 gcu_type8_c()                   | 123 |



---

|  |            |
|--|------------|
| 5.12.1.27 gcu_type8_i8()                 | 123        |
| 5.12.1.28 gcu_type8_ui8()                | 124        |
| 5.13 src/vector.c File Reference         | 124        |
| 5.14 test/test-debug.cpp File Reference  | 125        |
| 5.14.1 Detailed Description              | 126        |
| 5.15 test/test-hash.cpp File Reference   | 126        |
| 5.15.1 Detailed Description              | 127        |
| 5.16 test/test-string.cpp File Reference | 128        |
| 5.16.1 Detailed Description              | 128        |
| 5.17 test/test-type.cpp File Reference   | 128        |
| 5.17.1 Detailed Description              | 129        |
| 5.18 test/test-vector.cpp File Reference | 129        |
| 5.18.1 Detailed Description              | 130        |
| <b>Index</b>                             | <b>131</b> |



# 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:

|                                     |   |    |
|-------------------------------------|---|----|
| <a href="#">GCU_Hash16_Cell</a>     | 16-bit container holding the information for an entry in the hash table . . . . .   | 7  |
| <a href="#">GCU_Hash16_Iterator</a> | A container used to hold the state of an iterator which can be used to traverse all elements of a hash table . . . . .        | 8  |
| <a href="#">GCU_Hash16_Table</a>    | Container holding the information of the hash table . . . . .   | 9  |
| <a href="#">GCU_Hash16_Value</a>    | 16-bit container used to return the result of looking for a hash in the hash table . . . . .                                  | 10 |
| <a href="#">GCU_Hash32_Cell</a>     | 32-bit container holding the information for an entry in the hash table . . . . .   | 11 |
| <a href="#">GCU_Hash32_Iterator</a> | A container used to hold the state of an iterator which can be used to traverse all elements of a hash table . . . . .        | 12 |
| <a href="#">GCU_Hash32_Table</a>    | 32-bit container holding the information of the hash table . . . . .  | 13 |
| <a href="#">GCU_Hash32_Value</a>    | 32-bit container used to return the result of looking for a hash in the hash table . . . . .                                  | 14 |
| <a href="#">GCU_Hash64_Cell</a>     | 64-bit container holding the information for an entry in the hash table . . . . .   | 15 |
| <a href="#">GCU_Hash64_Iterator</a> | A 64-bit container used to hold the state of an iterator which can be used to traverse all elements of a hash table . . . . . | 16 |
| <a href="#">GCU_Hash64_Table</a>    | 64-bit container holding the information of the hash table . . . . .  | 17 |
| <a href="#">GCU_Hash64_Value</a>    | 64-bit container used to return the result of looking for a hash in the hash table . . . . .                                  | 18 |
| <a href="#">GCU_Hash8_Cell</a>      | 8-bit container holding the information for an entry in the hash table . . . . .  | 19 |
| <a href="#">GCU_Hash8_Iterator</a>  | A container used to hold the state of an iterator which can be used to traverse all elements of a hash table . . . . .        | 20 |
| <a href="#">GCU_Hash8_Table</a>     | Container holding the information of the hash table . . . . .   | 21 |
| <a href="#">GCU_Hash8_Value</a>     | 8-bit container used to return the result of looking for a hash in the hash table . . . . .                                   | 22 |

|   |                    |
|---|--------------------|
| <a href="#">GCU_Type16_Union</a>  |                    |
| A union of all basic, 16-bit types to be used by generic, 16-bit containers . . . . . | <a href="#">23</a> |
| <a href="#">GCU_Type32_Union</a>  |                    |
| A union of all basic, 32-bit types to be used by generic, 32-bit containers . . . . . | <a href="#">23</a> |
| <a href="#">GCU_Type64_Union</a>  |                    |
| A union of all basic, 64-bit types to be used by generic, 64-bit containers . . . . . | <a href="#">24</a> |
| <a href="#">GCU_Type8_Union</a>   |                    |
| A union of all basic, 8-bit types to be used by generic, 8-bit containers . . . . .   | <a href="#">24</a> |
| <a href="#">GCU_Vector16</a>  |                    |
| Container holding the information of the 16-bit vector . . . . .                      | <a href="#">25</a> |
| <a href="#">GCU_Vector32</a>  |                    |
| Container holding the information of the 32-bit vector . . . . .                      | <a href="#">26</a> |
| <a href="#">GCU_Vector64</a>  |                    |
| Container holding the information of the 64-bit vector . . . . .                      | <a href="#">27</a> |
| <a href="#">GCU_Vector8</a>   |                    |
| Container holding the information of the 8-bit vector . . . . .                       | <a href="#">28</a> |

## Chapter 3

# File Index

### 3.1 File List

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

|   |     |
|---|-----|
| <a href="#">include/cutil/debug.h</a>   |     |
| Header file for debugging-related functions   | 29  |
| <a href="#">include/cutil/float.h</a>   |     |
| Type definitions for float types  | 32  |
| <a href="#">include/cutil/hash.h</a>  |     |
| A simple hash table implementation  | 33  |
| <a href="#">include/cutil/libver.h</a>  |     |
| Header file used to control the version numbering and function namespace for all of the library | 51  |
| <a href="#">include/cutil/string.h</a>  |     |
| A collection of string-related functions  | 56  |
| <a href="#">include/cutil/type.h</a>  |     |
| Type definitions and utilities for use by the Ghoti.io projects                                 | 60  |
| <a href="#">include/cutil/vector.h</a>  |     |
| A simple vector implementation  | 91  |
| <a href="#">src/debug.c</a>   | 99  |
| <a href="#">src/float_identifier.c</a>  |     |
| Simple program to generate correct floating point type names for a given byte size              | 102 |
| <a href="#">src/hash.c</a>  | 103 |
| <a href="#">src/string.c</a>  | 103 |
| <a href="#">src/type.c</a>  | 108 |
| <a href="#">src/vector.c</a>  | 124 |
| <a href="#">test/test-debug.cpp</a>   |     |
| Test the behavior of Ghoti.io CUtil debug library and tools                                     | 125 |
| <a href="#">test/test-hash.cpp</a>  |     |
| Test the behavior of Ghoti.io CUtil hash table library  | 126 |
| <a href="#">test/test-string.cpp</a>  |     |
| Test the behavior of Ghoti.io CUtil string library  | 128 |
| <a href="#">test/test-type.cpp</a>  |     |
| Test the behavior of Ghoti.io CUtil type library  | 128 |
| <a href="#">test/test-vector.cpp</a>  |     |
| Test the behavior of Ghoti.io CUtil hash table library  | 129 |





## Chapter 4

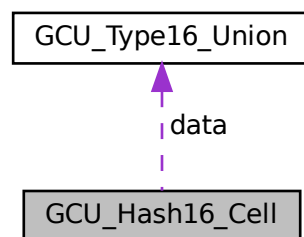
# Class Documentation

### 4.1 GCU\_Hash16\_Cell Struct Reference

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

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

Collaboration diagram for GCU\_Hash16\_Cell:



#### Public Attributes

- `size_t` `hash`  
*The hash of the entry.*
- `GCU_Type16_Union` `data`  
*The data of the entry.*
- `bool` `occupied`  
*Whether or not the entry has been initialized in some way.*
- `bool` `removed`  
*Whether or not the entry has been removed.*

### 4.1.1 Detailed Description

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

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

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

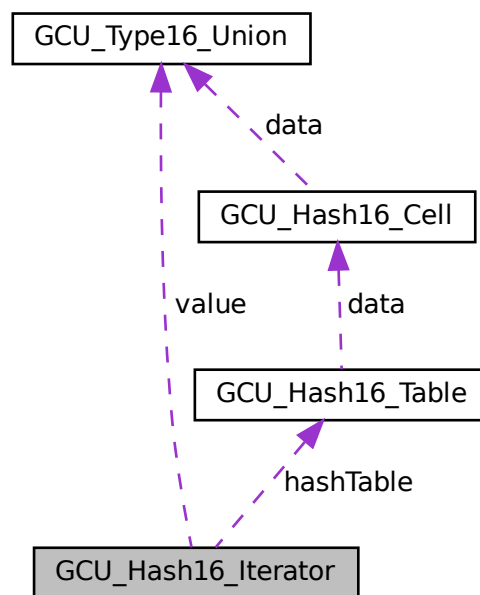
- `include/cutil/hash.h`

## 4.2 GCU\_Hash16\_Iterator Struct Reference

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

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

Collaboration diagram for GCU\_Hash16\_Iterator:



### Public Attributes

- `size_t current`  
The current index into the `hashTable` data structure corresponding to the iterator.
- `bool exists`  
Whether or not the iterator points to valid data.
- `GCU_Type16_Union value`  
The data pointed to by the iterator.
- `GCU_Hash16_Table * hashTable`  
The hash table that the iterator traverses.

### 4.2.1 Detailed Description

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

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

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

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

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

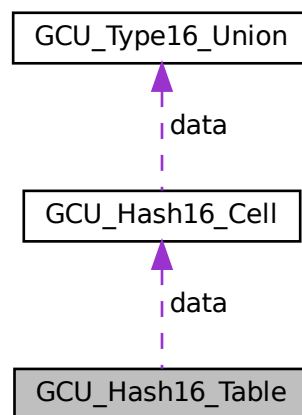
- `include/cutil/hash.h`

## 4.3 GCU\_Hash16\_Table Struct Reference

Container holding the information of the hash table.

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

Collaboration diagram for GCU\_Hash16\_Table:



### Public Attributes

- `size_t capacity`  
*The total item capacity of the hash table.*
- `size_t entries`  
*The count of non-empty cells.*
- `size_t removed`  
*The count of non-empty cells that represent elements which have been removed.*
- `GCU_Hash16_Cell * data`  
*A pointer to the array of data cells.*

### 4.3.1 Detailed Description

Container holding the information of the hash table.

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

1. Initialize the hash table using [gcu\\_hash16\\_create\(\)](#).
2. Destroy the has table using [gcu\\_hash16\\_destory\(\)](#).
3. Implementation of any thread-safety synchronization.
4. Life cycle management of the contents of the hash table. The hash table will **not**, for example, attempt to manage any pointers that it may contain upon deletion. The programmer is responsible for all memory management.

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

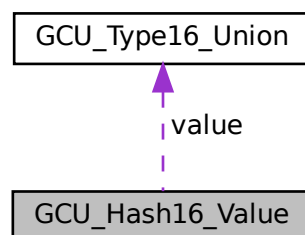
- [include/cutil/hash.h](#)

## 4.4 GCU\_Hash16\_Value Struct Reference

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

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

Collaboration diagram for GCU\_Hash16\_Value:



### Public Attributes

- `bool` [exists](#)  
*Whether or not the value exists in the hash table.*
- [GCU\\_Type16\\_Union](#) `value`  
*The value found in the table (if it exists).*

### 4.4.1 Detailed Description

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

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

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

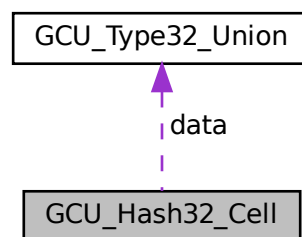
- `include/cutil/hash.h`

## 4.5 GCU\_Hash32\_Cell Struct Reference

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

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

Collaboration diagram for GCU\_Hash32\_Cell:



### Public Attributes

- `size_t hash`  
*The hash of the entry.*
- `GCU_Type32_Union data`  
*The data of the entry.*
- `bool occupied`  
*Whether or not the entry has been initialized in some way.*
- `bool removed`  
*Whether or not the entry has been removed.*

### 4.5.1 Detailed Description

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

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

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

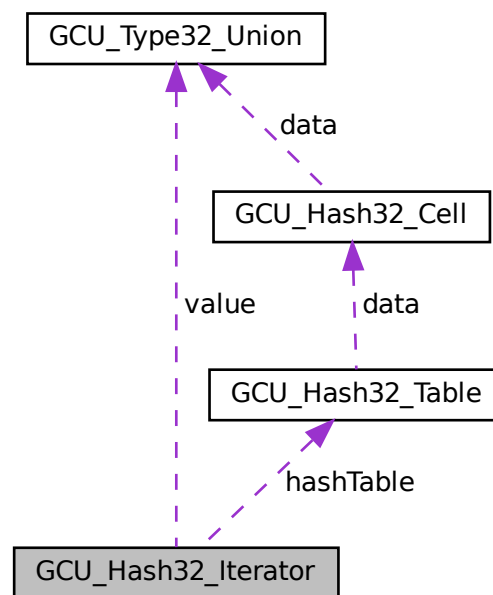
- `include/cutil/hash.h`

## 4.6 GCU\_Hash32\_Iterator Struct Reference

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

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

Collaboration diagram for GCU\_Hash32\_Iterator:



### Public Attributes

- `size_t current`  
The current index into the `hashTable` data structure corresponding to the iterator.
- `bool exists`  
Whether or not the iterator points to valid data.
- `GCU_Type32_Union value`  
The data pointed to by the iterator.
- `GCU_Hash32_Table * hashTable`  
The hash table that the iterator traverses.

### 4.6.1 Detailed Description

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

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

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

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

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

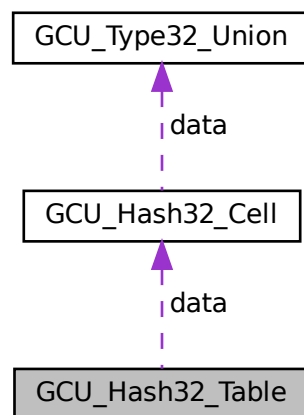
- `include/cutil/hash.h`

## 4.7 GCU\_Hash32\_Table Struct Reference

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

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

Collaboration diagram for GCU\_Hash32\_Table:



### Public Attributes

- `size_t` `capacity`  
*The total item capacity of the hash table.*
- `size_t` `entries`  
*The count of non-empty cells.*
- `size_t` `removed`  
*The count of non-empty cells that represent elements which have been removed.*
- `GCU_Hash32_Cell *` `data`  
*A pointer to the array of data cells.*

### 4.7.1 Detailed Description

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

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

1. Initialize the hash table using [gcu\\_hash32\\_create\(\)](#).
2. Destroy the has table using [gcu\\_hash32\\_destory\(\)](#).
3. Implementation of any thread-safety synchronization.
4. Life cycle management of the contents of the hash table. The hash table will **not**, for example, attempt to manage any pointers that it may contain upon deletion. The programmer is responsible for all memory management.

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

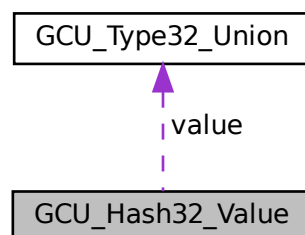
- [include/cutil/hash.h](#)

## 4.8 GCU\_Hash32\_Value Struct Reference

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

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

Collaboration diagram for GCU\_Hash32\_Value:



### Public Attributes

- `bool` [exists](#)  
*Whether or not the value exists in the hash table.*
- [GCU\\_Type32\\_Union](#) `value`  
*The value found in the table (if it exists).*



### 4.8.1 Detailed Description

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

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

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

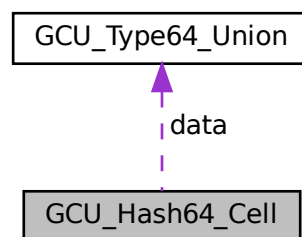
- `include/cutil/hash.h`

## 4.9 GCU\_Hash64\_Cell Struct Reference

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

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

Collaboration diagram for GCU\_Hash64\_Cell:



### Public Attributes

- `size_t hash`  
*The hash of the entry.*
- `GCU_Type64_Union data`  
*The data of the entry.*
- `bool occupied`  
*Whether or not the entry has been initialized in some way.*
- `bool removed`  
*Whether or not the entry has been removed.*

### 4.9.1 Detailed Description

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

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

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

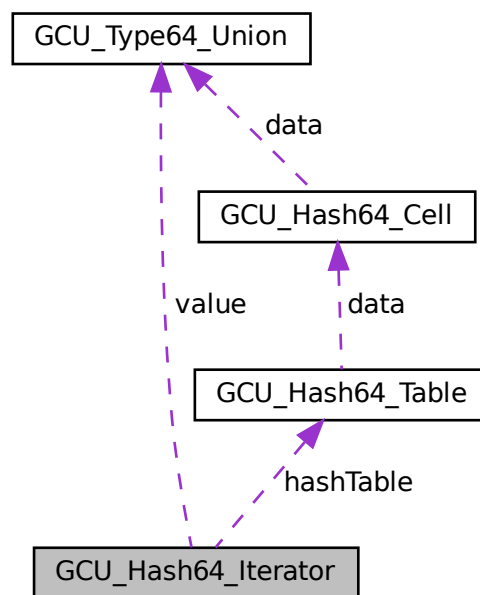
- `include/cutil/hash.h`

### 4.10 GCU\_Hash64\_Iterator Struct Reference

A 64-bit container used to hold the state of an iterator which can be used to traverse all elements of a hash table.

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

Collaboration diagram for `GCU_Hash64_Iterator`:



### Public Attributes

- `size_t current`  
The current index into the `hashTable` data structure corresponding to the iterator.
- `bool exists`  
Whether or not the iterator points to valid data.
- `GCU_Type64_Union value`  
The data pointed to by the iterator.
- `GCU_Hash64_Table * hashTable`  
The hash table that the iterator traverses.

### 4.10.1 Detailed Description

A 64-bit container used to hold the state of an iterator which can be used to traverse all elements of a hash table.

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

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

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

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

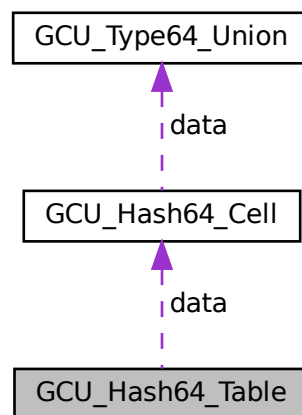
- `include/cutil/hash.h`

## 4.11 GCU\_Hash64\_Table Struct Reference

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

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

Collaboration diagram for GCU\_Hash64\_Table:



### Public Attributes

- `size_t capacity`  
*The total item capacity of the hash table.*
- `size_t entries`  
*The count of non-empty cells.*
- `size_t removed`  
*The count of non-empty cells that represent elements which have been removed.*
- `GCU_Hash64_Cell * data`  
*A pointer to the array of data cells.*

### 4.11.1 Detailed Description

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

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

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

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

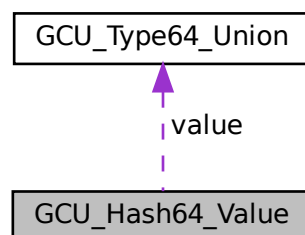
- `include/cutil/hash.h`

## 4.12 GCU\_Hash64\_Value Struct Reference

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

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

Collaboration diagram for GCU\_Hash64\_Value:



### Public Attributes

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

### 4.12.1 Detailed Description

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

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

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

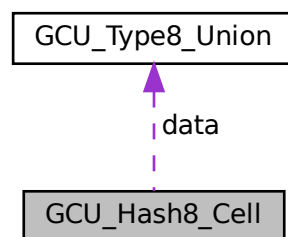
- `include/cutil/hash.h`

## 4.13 GCU\_Hash8\_Cell Struct Reference

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

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

Collaboration diagram for GCU\_Hash8\_Cell:



### Public Attributes

- `size_t hash`  
*The hash of the entry.*
- `GCU_Type8_Union data`  
*The data of the entry.*
- `bool occupied`  
*Whether or not the entry has been initialized in some way.*
- `bool removed`  
*Whether or not the entry has been removed.*

### 4.13.1 Detailed Description

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

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

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

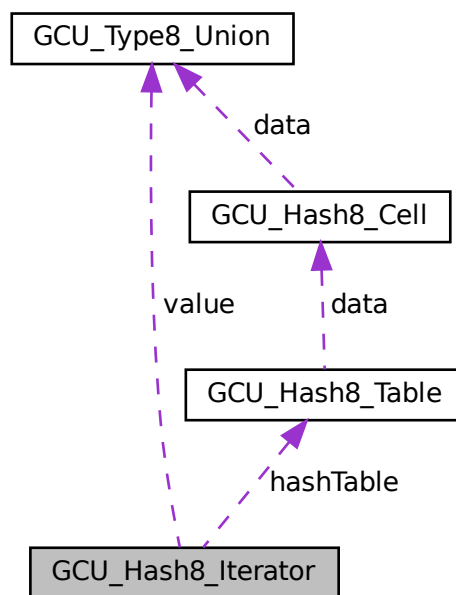
- `include/cutil/hash.h`

## 4.14 GCU\_Hash8\_Iterator Struct Reference

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

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

Collaboration diagram for GCU\_Hash8\_Iterator:



### Public Attributes

- `size_t current`  
The current index into the `hashTable` data structure corresponding to the iterator.
- `bool exists`  
Whether or not the iterator points to valid data.
- `GCU_Type8_Union value`  
The data pointed to by the iterator.
- `GCU_Hash8_Table * hashTable`  
The hash table that the iterator traverses.

### 4.14.1 Detailed Description

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

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

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

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

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

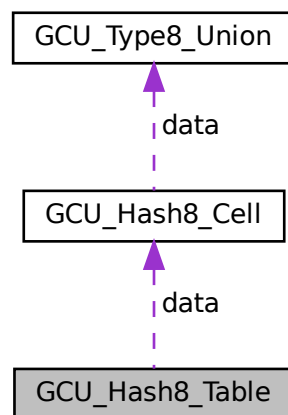
- `include/cutil/hash.h`

## 4.15 GCU\_Hash8\_Table Struct Reference

Container holding the information of the hash table.

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

Collaboration diagram for GCU\_Hash8\_Table:



### Public Attributes

- `size_t capacity`  
*The total item capacity of the hash table.*
- `size_t entries`  
*The count of non-empty cells.*
- `size_t removed`  
*The count of non-empty cells that represent elements which have been removed.*
- `GCU_Hash8_Cell * data`  
*A pointer to the array of data cells.*

### 4.15.1 Detailed Description

Container holding the information of the hash table.

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

1. Initialize the hash table using [gcu\\_hash8\\_create\(\)](#).
2. Destroy the has table using [gcu\\_hash8\\_destory\(\)](#).
3. Implementation of any thread-safety synchronization.
4. Life cycle management of the contents of the hash table. The hash table will **not**, for example, attempt to manage any pointers that it may contain upon deletion. The programmer is responsible for all memory management.

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

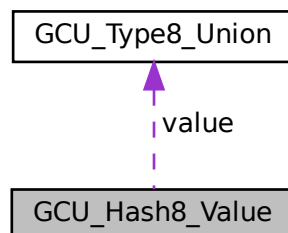
- [include/cutil/hash.h](#)

## 4.16 GCU\_Hash8\_Value Struct Reference

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

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

Collaboration diagram for GCU\_Hash8\_Value:



### Public Attributes

- [bool exists](#)  
*Whether or not the value exists in the hash table.*
- [GCU\\_Type8\\_Union value](#)  
*The value found in the table (if it exists).*



### 4.16.1 Detailed Description

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

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

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

- `include/cutil/hash.h`

## 4.17 GCU\_Type16\_Union Union Reference

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

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

### Public Attributes

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

### 4.17.1 Detailed Description

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

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

- `include/cutil/type.h`

## 4.18 GCU\_Type32\_Union Union Reference

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

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

### Public Attributes

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

### 4.18.1 Detailed Description

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

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

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

## 4.19 GCU\_Type64\_Union Union Reference

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

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

### Public Attributes

- void \* **p**
- uint64\_t **ui64**
- uint32\_t **ui32**
- uint16\_t **ui16**
- uint8\_t **ui8**
- int64\_t **i64**
- int32\_t **i32**
- int16\_t **i16**
- int8\_t **i8**
- GCU\_float64\_t **f64**
- GCU\_float32\_t **f32**
- char **c**

### 4.19.1 Detailed Description

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

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

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

## 4.20 GCU\_Type8\_Union Union Reference

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

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

### Public Attributes

- uint8\_t **ui8**
- int8\_t **i8**
- char **c**

### 4.20.1 Detailed Description

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

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

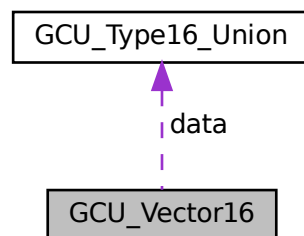
- [include/cutil/type.h](#)

## 4.21 GCU\_Vector16 Struct Reference

Container holding the information of the 16-bit vector.

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

Collaboration diagram for GCU\_Vector16:



### Public Attributes

- `size_t` [capacity](#)  
*The total item capacity of the vector.*
- `size_t` [count](#)  
*The count of non-empty cells.*
- [GCU\\_Type16\\_Union](#) \* [data](#)  
*A pointer to the array of data cells.*

### 4.21.1 Detailed Description

Container holding the information of the 16-bit vector.

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

1. Initialize the vector using [gcu\\_vector16\\_create\(\)](#).
2. Destroy the vector using [gcu\\_vector16\\_destroy\(\)](#).
3. Implementation of any thread-safety synchronization.
4. Life cycle management of the contents of the vector. The vector will **not**, for example, attempt to manage any pointers that it may contain upon deletion. The programmer is responsible for all memory management.

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

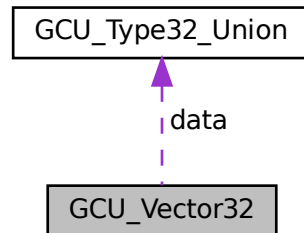
- [include/cutil/vector.h](#)

## 4.22 GCU\_Vector32 Struct Reference

Container holding the information of the 32-bit vector.

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

Collaboration diagram for GCU\_Vector32:



### Public Attributes

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

#### 4.22.1 Detailed Description

Container holding the information of the 32-bit vector.

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

1. Initialize the vector using [gcu\\_vector32\\_create\(\)](#).
2. Destroy the vector using [gcu\\_vector32\\_destroy\(\)](#).
3. Implementation of any thread-safety synchronization.
4. Life cycle management of the contents of the vector. The vector will **not**, for example, attempt to manage any pointers that it may contain upon deletion. The programmer is responsible for all memory management.

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

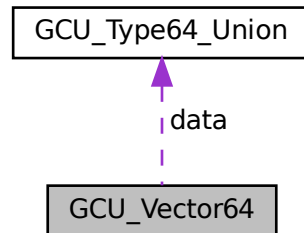
- `include/cutil/`[vector.h](#)

## 4.23 GCU\_Vector64 Struct Reference

Container holding the information of the 64-bit vector.

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

Collaboration diagram for GCU\_Vector64:



### Public Attributes

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

#### 4.23.1 Detailed Description

Container holding the information of the 64-bit vector.

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

1. Initialize the vector using `gcu_vector64_create()`.
2. Destroy the vector using `gcu_vector64_destroy()`.
3. Implementation of any thread-safety synchronization.
4. Life cycle management of the contents of the vector. The vector will **not**, for example, attempt to manage any pointers that it may contain upon deletion. The programmer is responsible for all memory management.

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

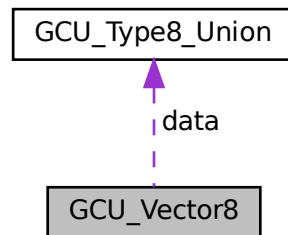
- `include/cutil/vector.h`

## 4.24 GCU\_Vector8 Struct Reference

Container holding the information of the 8-bit vector.

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

Collaboration diagram for GCU\_Vector8:



### Public Attributes

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

### 4.24.1 Detailed Description

Container holding the information of the 8-bit vector.

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

1. Initialize the vector using [gcu\\_vector8\\_create\(\)](#).
2. Destroy the vector using [gcu\\_vector8\\_destroy\(\)](#).
3. Implementation of any thread-safety synchronization.
4. Life cycle management of the contents of the vector. The vector will **not**, for example, attempt to manage any pointers that it may contain upon deletion. The programmer is responsible for all memory management.

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

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

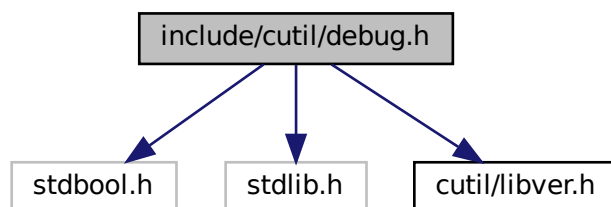
## Chapter 5

# File Documentation

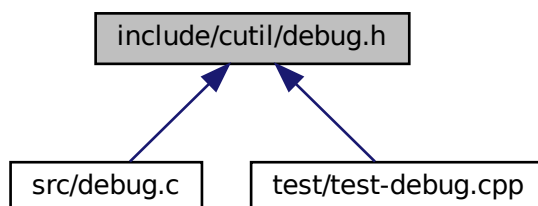
### 5.1 include/cutil/debug.h File Reference

Header file for debugging-related functions.

```
#include <stdbool.h>
#include <stdlib.h>
#include "cutil/libver.h"
Include dependency graph for debug.h:
```



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



## Functions

- void \* [gcu\\_malloc](#) (size\_t size, const char \*file, size\_t line)  
*Wrapper for the standard malloc() function.*
- void \* [gcu\\_calloc](#) (size\_t nitems, size\_t size, const char \*file, size\_t line)  
*Wrapper for the standard calloc() function.*
- void \* [gcu\\_realloc](#) (void \*pointer, size\_t size, const char \*file, size\_t line)  
*Wrapper for the standard realloc() function.*
- void [gcu\\_free](#) (void \*pointer, const char \*file, size\_t line)  
*Wrapper for the standard free() function.*
- void [gcu\\_mem\\_start](#) (void)  
*Signal that intercepted memory management calls should be logged to stderr.*
- void [gcu\\_mem\\_stop](#) (void)  
*Signal that intercepted memory management calls should no longer be logged to stderr.*

### 5.1.1 Detailed Description

Header file for debugging-related functions.

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

Logging to `stderr` is enabled by default. It may be disabled by calling [gcu\\_mem\\_stop\(\)](#), and re-enabled by calling [gcu\\_mem\\_start\(\)](#).

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

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

### 5.1.2 Function Documentation

#### 5.1.2.1 gcu\_calloc()

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

Wrapper for the standard `calloc()` function.

#### Parameters

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



**Returns**

The beginning byte of the allocated memory.

**5.1.2.2 gcu\_free()**

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

Wrapper for the standard free() function.

**Parameters**

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

**5.1.2.3 gcu\_malloc()**

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

Wrapper for the standard malloc() function.

**Parameters**

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

**Returns**

The beginning byte of the allocated memory.

**5.1.2.4 gcu\_realloc()**

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

```

size_t size,
const char * file,
size_t line )

```

Wrapper for the standard `realloc()` function.

#### Parameters

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

#### Returns

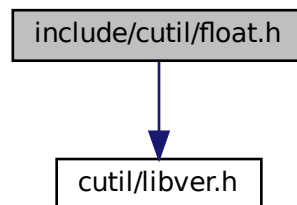
The beginning byte of the reallocated memory.

## 5.2 include/cutil/float.h File Reference

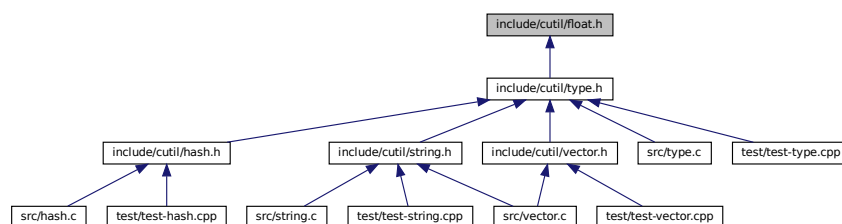
Type definitions for float types.

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

Include dependency graph for float.h:



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



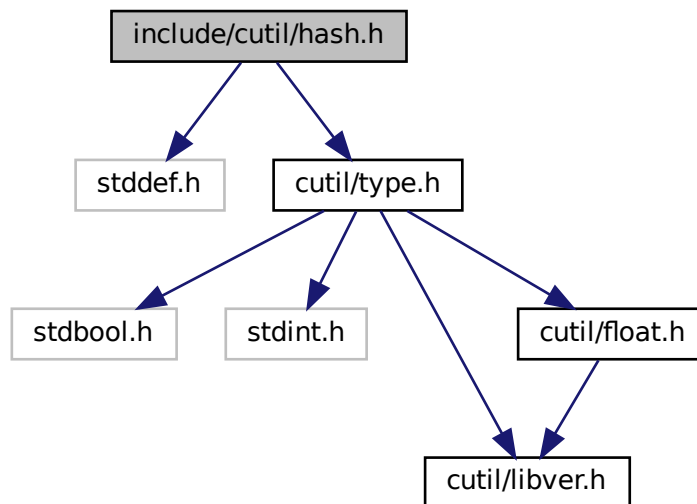
### 5.2.1 Detailed Description

Type definitions for float types.

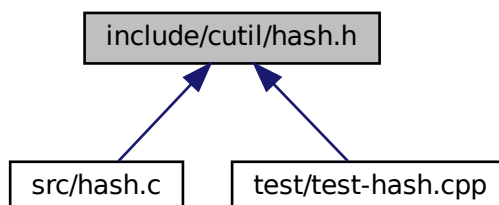
## 5.3 include/cutil/hash.h File Reference

A simple hash table implementation.

```
#include <stddef.h>
#include "cutil/type.h"
Include dependency graph for hash.h:
```



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



## Classes

- struct [GCU\\_Hash64\\_Value](#)  
*64-bit container used to return the result of looking for a hash in the hash table.*
- struct [GCU\\_Hash64\\_Cell](#)  
*64-bit container holding the information for an entry in the hash table.*
- struct [GCU\\_Hash64\\_Table](#)  
*64-bit container holding the information of the hash table.*
- struct [GCU\\_Hash64\\_Iterator](#)  
*A 64-bit container used to hold the state of an iterator which can be used to traverse all elements of a hash table.*
- struct [GCU\\_Hash32\\_Value](#)  
*32-bit container used to return the result of looking for a hash in the hash table.*
- struct [GCU\\_Hash32\\_Cell](#)  
*32-bit container holding the information for an entry in the hash table.*
- struct [GCU\\_Hash32\\_Table](#)  
*32-bit container holding the information of the hash table.*
- struct [GCU\\_Hash32\\_Iterator](#)  
*A container used to hold the state of an iterator which can be used to traverse all elements of a hash table.*
- struct [GCU\\_Hash16\\_Value](#)  
*16-bit container used to return the result of looking for a hash in the hash table.*
- struct [GCU\\_Hash16\\_Cell](#)  
*16-bit container holding the information for an entry in the hash table.*
- struct [GCU\\_Hash16\\_Table](#)  
*Container holding the information of the hash table.*
- struct [GCU\\_Hash16\\_Iterator](#)  
*A container used to hold the state of an iterator which can be used to traverse all elements of a hash table.*
- struct [GCU\\_Hash8\\_Value](#)  
*8-bit container used to return the result of looking for a hash in the hash table.*
- struct [GCU\\_Hash8\\_Cell](#)  
*8-bit container holding the information for an entry in the hash table.*
- struct [GCU\\_Hash8\\_Table](#)  
*Container holding the information of the hash table.*
- struct [GCU\\_Hash8\\_Iterator](#)  
*A container used to hold the state of an iterator which can be used to traverse all elements of a hash table.*

## Functions

- [GCU\\_Hash64\\_Table](#) \* [gcu\\_hash64\\_create](#) (size\_t count)  
*Create a hash table structure for 64-bit entries.*
- void [gcu\\_hash64\\_destroy](#) ([GCU\\_Hash64\\_Table](#) \*hashTable)  
*Destroy a hash table structure and clean up memory allocations.*
- bool [gcu\\_hash64\\_set](#) ([GCU\\_Hash64\\_Table](#) \*hashTable, size\_t hash, [GCU\\_Type64\\_Union](#) value)  
*Set a value in the hash table.*
- [GCU\\_Hash64\\_Value](#) [gcu\\_hash64\\_get](#) ([GCU\\_Hash64\\_Table](#) \*hashTable, size\_t hash)  
*Get a value from the hash table (if it exists).*
- bool [gcu\\_hash64\\_contains](#) ([GCU\\_Hash64\\_Table](#) \*hashTable, size\_t hash)  
*Check to see whether or not a hash table contains a specific hash.*
- bool [gcu\\_hash64\\_remove](#) ([GCU\\_Hash64\\_Table](#) \*hashTable, size\_t hash)  
*Remove a hash from the table.*
- size\_t [gcu\\_hash64\\_count](#) ([GCU\\_Hash64\\_Table](#) \*hashTable)

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

  - [GCU\\_Hash64\\_Iterator](#) [gcu\\_hash64\\_iterator\\_get](#) ([GCU\\_Hash64\\_Table](#) \*hashTable)

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

  - [GCU\\_Hash64\\_Iterator](#) [gcu\\_hash64\\_iterator\\_next](#) ([GCU\\_Hash64\\_Iterator](#) iterator)

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

  - [GCU\\_Hash32\\_Table](#) \* [gcu\\_hash32\\_create](#) (size\_t count)

*Create a hash table structure for 32-bit entries.*

  - void [gcu\\_hash32\\_destroy](#) ([GCU\\_Hash32\\_Table](#) \*hashTable)

*Destroy a hash table structure and clean up memory allocations.*

  - bool [gcu\\_hash32\\_set](#) ([GCU\\_Hash32\\_Table](#) \*hashTable, size\_t hash, [GCU\\_Type32\\_Union](#) value)

*Set a value in the hash table.*

  - [GCU\\_Hash32\\_Value](#) [gcu\\_hash32\\_get](#) ([GCU\\_Hash32\\_Table](#) \*hashTable, size\_t hash)

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

  - bool [gcu\\_hash32\\_contains](#) ([GCU\\_Hash32\\_Table](#) \*hashTable, size\_t hash)

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

  - bool [gcu\\_hash32\\_remove](#) ([GCU\\_Hash32\\_Table](#) \*hashTable, size\_t hash)

*Remove a hash from the table.*

  - size\_t [gcu\\_hash32\\_count](#) ([GCU\\_Hash32\\_Table](#) \*hashTable)

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

  - [GCU\\_Hash32\\_Iterator](#) [gcu\\_hash32\\_iterator\\_get](#) ([GCU\\_Hash32\\_Table](#) \*hashTable)

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

  - [GCU\\_Hash32\\_Iterator](#) [gcu\\_hash32\\_iterator\\_next](#) ([GCU\\_Hash32\\_Iterator](#) iterator)

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

  - [GCU\\_Hash16\\_Table](#) \* [gcu\\_hash16\\_create](#) (size\_t count)

*Create a hash table structure.*

  - void [gcu\\_hash16\\_destroy](#) ([GCU\\_Hash16\\_Table](#) \*hashTable)

*Destroy a hash table structure and clean up memory allocations.*

  - bool [gcu\\_hash16\\_set](#) ([GCU\\_Hash16\\_Table](#) \*hashTable, size\_t hash, [GCU\\_Type16\\_Union](#) value)

*Set a value in the hash table.*

  - [GCU\\_Hash16\\_Value](#) [gcu\\_hash16\\_get](#) ([GCU\\_Hash16\\_Table](#) \*hashTable, size\_t hash)

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

  - bool [gcu\\_hash16\\_contains](#) ([GCU\\_Hash16\\_Table](#) \*hashTable, size\_t hash)

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

  - bool [gcu\\_hash16\\_remove](#) ([GCU\\_Hash16\\_Table](#) \*hashTable, size\_t hash)

*Remove a hash from the table.*

  - size\_t [gcu\\_hash16\\_count](#) ([GCU\\_Hash16\\_Table](#) \*hashTable)

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

  - [GCU\\_Hash16\\_Iterator](#) [gcu\\_hash16\\_iterator\\_get](#) ([GCU\\_Hash16\\_Table](#) \*hashTable)

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

  - [GCU\\_Hash16\\_Iterator](#) [gcu\\_hash16\\_iterator\\_next](#) ([GCU\\_Hash16\\_Iterator](#) iterator)

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

  - [GCU\\_Hash8\\_Table](#) \* [gcu\\_hash8\\_create](#) (size\_t count)

*Create a hash table structure.*

  - void [gcu\\_hash8\\_destroy](#) ([GCU\\_Hash8\\_Table](#) \*hashTable)

*Destroy a hash table structure and clean up memory allocations.*

  - bool [gcu\\_hash8\\_set](#) ([GCU\\_Hash8\\_Table](#) \*hashTable, size\_t hash, [GCU\\_Type8\\_Union](#) value)

*Set a value in the hash table.*

  - [GCU\\_Hash8\\_Value](#) [gcu\\_hash8\\_get](#) ([GCU\\_Hash8\\_Table](#) \*hashTable, size\_t hash)

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

  - bool [gcu\\_hash8\\_contains](#) ([GCU\\_Hash8\\_Table](#) \*hashTable, size\_t hash)

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

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

### 5.3.1 Detailed Description

A simple hash table implementation.

### 5.3.2 Function Documentation

#### 5.3.2.1 gcu\_hash16\_contains()

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

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

##### Parameters

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

##### Returns

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

#### 5.3.2.2 gcu\_hash16\_count()

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

Get a count of active entries in the hash table.

##### Parameters

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

#### Returns

The count of active entries in the hash table.

#### 5.3.2.3 gcu\_hash16\_create()

```
GCU_Hash16_Table* gcu_hash16_create (
    size_t count )
```

Create a hash table structure.

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

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

#### Parameters

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

#### Returns

A struct containing the hash table information.

#### 5.3.2.4 gcu\_hash16\_destroy()

```
void gcu_hash16_destroy (
    GCU_Hash16_Table * hashTable )
```

Destroy a hash table structure and clean up memory allocations.

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

#### Parameters

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

#### 5.3.2.5 gcu\_hash16\_get()

```
GCU_Hash16_Value gcu_hash16_get (
```

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

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

#### Parameters

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

#### Returns

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

### 5.3.2.6 gcu\_hash16\_iterator\_get()

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

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

#### Parameters

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

#### Returns

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

### 5.3.2.7 gcu\_hash16\_iterator\_next()

```
GCU_Hash16_Iterator gcu_hash16_iterator_next (  
    GCU_Hash16_Iterator iterator )
```

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

Any change to the hash table (such as setting a value) might alter the underlying structure of the hash table, which would invalidate the iterator. Any call to [gcu\\_hash16\\_set\(\)](#), therefore, should be considered as an invalidation of any iterators associated with the hash table.

#### Parameters

|                 |  |
|-----------------|--|
| <i>iterator</i> | The iterator from which to calculate and return the next iterator. |
|-----------------|--|



#### Returns

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

#### 5.3.2.8 gcu\_hash16\_remove()

```
bool gcu_hash16_remove (
    GCU_Hash16_Table * hashTable,
    size_t hash )
```

Remove a hash from the table.

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

#### Parameters

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

#### Returns

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

#### 5.3.2.9 gcu\_hash16\_set()

```
bool gcu_hash16_set (
    GCU_Hash16_Table * hashTable,
    size_t hash,
    GCU_Type16_Union value )
```

Set a value in the hash table.

Setting a value may trigger a resize of the hash table. This can be avoided entirely by setting an appropriate `count` value when creating the hash table with [gcu\\_hash16\\_create\(\)](#).

#### Parameters

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

#### Returns

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

### 5.3.2.10 `gcu_hash32_contains()`

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

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

#### Parameters

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

#### Returns

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

### 5.3.2.11 `gcu_hash32_count()`

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

Get a count of active entries in the hash table.

#### Parameters

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

#### Returns

The count of active entries in the hash table.

### 5.3.2.12 `gcu_hash32_create()`

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

Create a hash table structure for 32-bit entries.

All invocations of a hash table must have a corresponding `gcu_hash32_destroy()` call in order to clean up dynamically-allocated memory.

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

## Parameters

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

## Returns

A struct containing the hash table information.

**5.3.2.13 gcu\_hash32\_destroy()**

```
void gcu_hash32_destroy (
    GCU_Hash32_Table * hashTable )
```

Destroy a hash table structure and clean up memory allocations.

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

## Parameters

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

**5.3.2.14 gcu\_hash32\_get()**

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

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

## Parameters

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

## Returns

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

**5.3.2.15 gcu\_hash32\_iterator\_get()**

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

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

## Parameters

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

## Returns

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

**5.3.2.16 gcu\_hash32\_iterator\_next()**

```
GCU_Hash32_Iterator gcu_hash32_iterator_next (
    GCU_Hash32_Iterator iterator )
```

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

Any change to the hash table (such as setting a value) might alter the underlying structure of the hash table, which would invalidate the iterator. Any call to [gcu\\_hash32\\_set\(\)](#), therefore, should be considered as an invalidation of any iterators associated with the hash table.

## Parameters

|                 |  |
|-----------------|--|
| <i>iterator</i> | The iterator from which to calculate and return the next iterator. |
|-----------------|--|

## Returns

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

**5.3.2.17 gcu\_hash32\_remove()**

```
bool gcu_hash32_remove (
    GCU_Hash32_Table * hashTable,
    size_t hash )
```

Remove a hash from the table.

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

## Parameters

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

**Returns**

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

**5.3.2.18 gcu\_hash32\_set()**

```
bool gcu_hash32_set (
    GCU_Hash32_Table * hashTable,
    size_t hash,
    GCU_Type32_Union value )
```

Set a value in the hash table.

Setting a value may trigger a resize of the hash table. This can be avoided entirely by setting an appropriate `count` value when creating the hash table with [gcu\\_hash32\\_create\(\)](#).

**Parameters**

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

**Returns**

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

**5.3.2.19 gcu\_hash64\_contains()**

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

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

**Parameters**

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

**Returns**

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

### 5.3.2.20 `gcu_hash64_count()`

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

Get a count of active entries in the hash table.

#### Parameters

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

#### Returns

The count of active entries in the hash table.

### 5.3.2.21 `gcu_hash64_create()`

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

Create a hash table structure for 64-bit entries.

All invocations of a hash table must have a corresponding `gcu_hash64_destroy()` call in order to clean up dynamically-allocated memory.

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

#### Parameters

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

#### Returns

A struct containing the hash table information.

### 5.3.2.22 `gcu_hash64_destroy()`

```
void gcu_hash64_destroy (
    GCU_Hash64_Table * hashTable )
```

Destroy a hash table structure and clean up memory allocations.

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

**Parameters**

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

**5.3.2.23 gcu\_hash64\_get()**

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

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

**Parameters**

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

**Returns**

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

**5.3.2.24 gcu\_hash64\_iterator\_get()**

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

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

**Parameters**

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

**Returns**

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

**5.3.2.25 gcu\_hash64\_iterator\_next()**

```
GCU_Hash64_Iterator gcu_hash64_iterator_next (
    GCU_Hash64_Iterator iterator )
```



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

Any change to the hash table (such as setting a value) might alter the underlying structure of the hash table, which would invalidate the iterator. Any call to `gcu_hash64_set()`, therefore, should be considered as an invalidation of any iterators associated with the hash table.

#### Parameters

|                 |  |
|-----------------|--|
| <i>iterator</i> | The iterator from which to calculate and return the next iterator. |
|-----------------|--|

#### Returns

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

### 5.3.2.26 `gcu_hash64_remove()`

```
bool gcu_hash64_remove (
    GCU_Hash64_Table * hashTable,
    size_t hash )
```

Remove a hash from the table.

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

#### Parameters

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

#### Returns

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

### 5.3.2.27 `gcu_hash64_set()`

```
bool gcu_hash64_set (
    GCU_Hash64_Table * hashTable,
    size_t hash,
    GCU_Type64_Union value )
```

Set a value in the hash table.

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

**Parameters**

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

**Returns**

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

**5.3.2.28 gcu\_hash8\_contains()**

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

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

**Parameters**

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

**Returns**

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

**5.3.2.29 gcu\_hash8\_count()**

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

Get a count of active entries in the hash table.

**Parameters**

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

**Returns**

The count of active entries in the hash table.

### 5.3.2.30 `gcu_hash8_create()`

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

Create a hash table structure.

All invocations of a hash table must have a corresponding `gcu_hash8_destroy()` call in order to clean up dynamically-allocated memory.

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

#### Parameters

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

#### Returns

A struct containing the hash table information.

### 5.3.2.31 `gcu_hash8_destroy()`

```
void gcu_hash8_destroy (
    GCU_Hash8_Table * hashTable )
```

Destroy a hash table structure and clean up memory allocations.

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

#### Parameters

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

### 5.3.2.32 `gcu_hash8_get()`

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

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

**Parameters**

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

**Returns**

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

**5.3.2.33 gcu\_hash8\_iterator\_get()**

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

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

**Parameters**

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

**Returns**

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

**5.3.2.34 gcu\_hash8\_iterator\_next()**

```
GCU_Hash8_Iterator gcu_hash8_iterator_next (
    GCU_Hash8_Iterator iterator )
```

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

Any change to the hash table (such as setting a value) might alter the underlying structure of the hash table, which would invalidate the iterator. Any call to [gcu\\_hash8\\_set\(\)](#), therefore, should be considered as an invalidation of any iterators associated with the hash table.

**Parameters**

|                 |  |
|-----------------|--|
| <i>iterator</i> | The iterator from which to calculate and return the next iterator. |
|-----------------|--|

**Returns**

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

### 5.3.2.35 gcu\_hash8\_remove()

```
bool gcu_hash8_remove (
    GCU_Hash8_Table * hashTable,
    size_t hash )
```

Remove a hash from the table.

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

#### Parameters

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

#### Returns

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

### 5.3.2.36 gcu\_hash8\_set()

```
bool gcu_hash8_set (
    GCU_Hash8_Table * hashTable,
    size_t hash,
    GCU_Type8_Union value )
```

Set a value in the hash table.

Setting a value may trigger a resize of the hash table. This can be avoided entirely by setting an appropriate `count` value when creating the hash table with [gcu\\_hash8\\_create\(\)](#).

#### Parameters

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

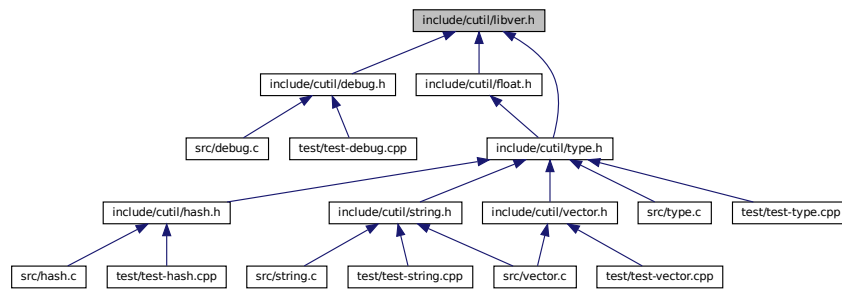
#### Returns

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

## 5.4 include/cutil/libver.h File Reference

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

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



## Macros

- `#define GHOTIIO_CUTIL_NAME ghotiio_cutil_dev`  
*Used in conjunction with the GHOTIIO\_CUTIL...*
- `#define GHOTIIO_CUTIL_VERSION "dev"`  
*String representation of the version, provided as a convenience to the programmer.*
- `#define GHOTIIO_CUTIL(NAME) GHOTIIO_CUTIL_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 #defines properly.*
- `#define GHOTIIO_CUTIL_RENAME(a, b) GHOTIIO_CUTIL_RENAME_INNER(a,b)`  
*Helper macro to concatenate the #defines 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.*

### 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

|             |   |
|-------------|---|
| <i>NAME</i> | 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

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

## Returns

A call to the [GHOTIIO\\_CUTIL\\_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 requires two levels of processing.

This macro should not be called directly. It should only be called by [GHOTIIO\\_CUTIL\\_CONCAT2\(\)](#).

## Parameters

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

## Returns

The concatenation of *a* to *b*.

#### 5.4.2.4 GHOTIIO\_CUTIL\_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

|          |                                    |
|----------|------------------------------------|
| <i>a</i> | The first part of the identifier.  |
| <i>b</i> | The second part of the identifier. |
| <i>c</i> | 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 requires two levels of processing.

This macro should not be called directly. It should only be called by [GHOTIIO\\_CUTIL\\_CONCAT2\(\)](#).

##### Parameters

|          |                                    |
|----------|------------------------------------|
| <i>a</i> | The first part of the identifier.  |
| <i>b</i> | The second part of the identifier. |
| <i>c</i> | 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 #defines properly.

It requires two levels of processing.

##### Parameters

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

##### Returns

A call to the `GHOTIIO_CUTIL_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 #defines properly.

It requires two levels of processing.

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

##### Parameters

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

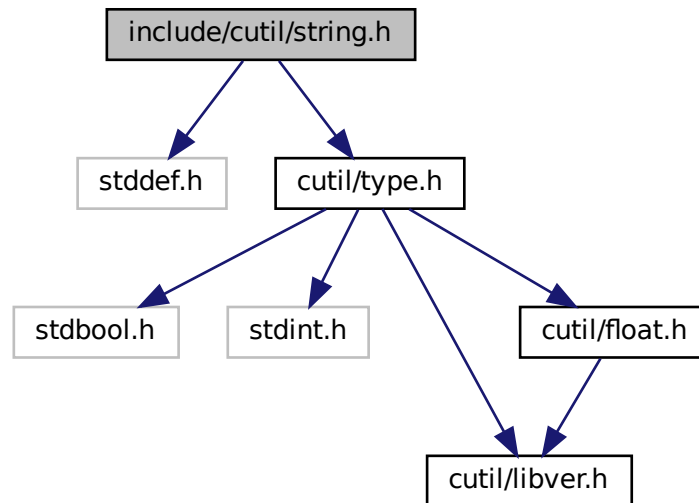
##### Returns

The concatenation of *a* to *b*.

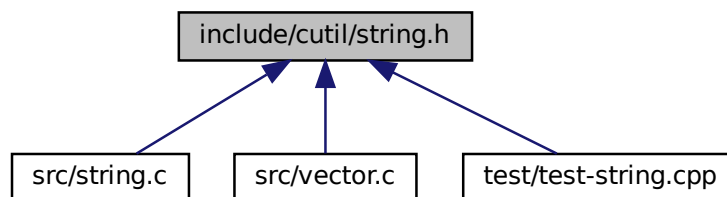
## 5.5 include/cutil/string.h File Reference

A collection of string-related functions.

```
#include <stddef.h>
#include "cutil/type.h"
Include dependency graph for string.h:
```



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



## 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.*

### 5.5.1 Detailed Description

A collection of string-related functions.

### 5.5.2 Function Documentation

#### 5.5.2.1 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

|            |  |
|------------|--|
| <i>str</i> | A pointer to the string (or data block). |
| <i>len</i> | The length of the data in bytes.         |

##### Returns

A 32-bit number representing the value.

Here is the call graph for this function:



### 5.5.2.2 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.

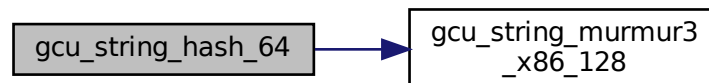
#### Parameters

|            |  |
|------------|--|
| <i>str</i> | A pointer to the string (or data block). |
| <i>len</i> | The length of the data in bytes.         |

#### Returns

A 64-bit number representing the value.

Here is the call graph for this function:



### 5.5.2.3 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

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

#### 5.5.2.4 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

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

#### 5.5.2.5 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>

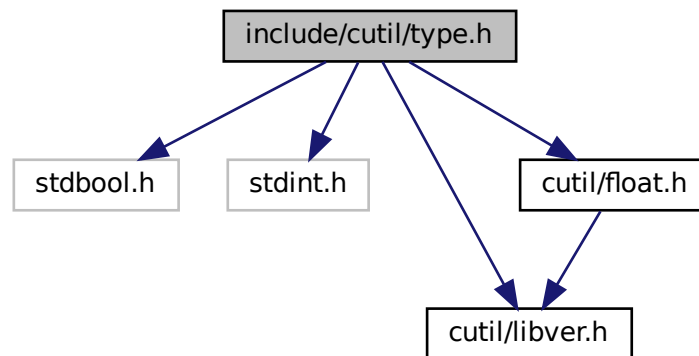
##### Parameters

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

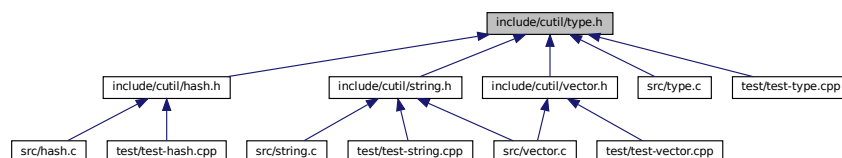
## 5.6 include/cutil/type.h File Reference

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\\_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\\_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_c (char val)`

Create a 64-bit union variable with the type *char*.
- `GCU_Type32_Union gcu_type32_ui32 (uint32_t val)`

Create a 32-bit union variable with the type *uint32\_t*.
- `GCU_Type32_Union gcu_type32_ui16 (uint16_t val)`

Create a 32-bit union variable with the type *uint16\_t*.
- `GCU_Type32_Union gcu_type32_ui8 (uint8_t val)`

Create a 32-bit union variable with the type *uint8\_t*.
- `GCU_Type32_Union gcu_type32_i32 (int32_t val)`

Create a 32-bit union variable with the type *int32\_t*.
- `GCU_Type32_Union gcu_type32_i16 (int16_t val)`

Create a 32-bit union variable with the type *int16\_t*.
- `GCU_Type32_Union gcu_type32_i8 (int8_t val)`

Create a 32-bit union variable with the type *int8\_t*.
- `GCU_Type32_Union gcu_type32_f32 (GCU_float32_t val)`

Create a 32-bit union variable with the type *float* with 32 bits.
- `GCU_Type32_Union gcu_type32_c (char val)`

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



- [GCU\\_Type16\\_Union gcu\\_type16\\_ui16](#) (uint16\_t val)  
*Create a 16-bit union variable with the type uint16\_t.*
- [GCU\\_Type16\\_Union gcu\\_type16\\_ui8](#) (uint8\_t val)  
*Create a 16-bit union variable with the type uint8\_t.*
- [GCU\\_Type16\\_Union gcu\\_type16\\_i16](#) (int16\_t val)  
*Create a 16-bit union variable with the type int16\_t.*
- [GCU\\_Type16\\_Union gcu\\_type16\\_i8](#) (int8\_t val)  
*Create a 16-bit union variable with the type int8\_t.*
- [GCU\\_Type16\\_Union gcu\\_type16\\_c](#) (char val)  
*Create a 16-bit union variable with the type char.*
- [GCU\\_Type8\\_Union gcu\\_type8\\_ui8](#) (uint8\_t val)  
*Create a 8-bit union variable with the type uint8\_t.*
- [GCU\\_Type8\\_Union gcu\\_type8\\_i8](#) (int8\_t val)  
*Create a 8-bit union variable with the type int8\_t.*
- [GCU\\_Type8\\_Union gcu\\_type8\\_c](#) (char val)  
*Create a 8-bit union variable with the type char.*

### 5.6.1 Detailed Description

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

### 5.6.2 Macro Definition Documentation

#### 5.6.2.1 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

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

Returns

The union variable.

### 5.6.2.2 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

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

#### Returns

The union variable.

### 5.6.2.3 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

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

#### Returns

The union variable.

### 5.6.2.4 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

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

#### Returns

The union variable.

### 5.6.2.5 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

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

#### Returns

The union variable.

### 5.6.2.6 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

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

Returns

The union variable.

### 5.6.2.7 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

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

Returns

The union variable.

### 5.6.2.8 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

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

#### Returns

The union variable.

### 5.6.2.9 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

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

#### Returns

The union variable.

### 5.6.2.10 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**

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

**Returns**

The union variable.

**5.6.2.11 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**

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

**Returns**

The union variable.

**5.6.2.12 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**

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

**Returns**

The union variable.

**5.6.2.13 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**

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

**Returns**

The union variable.

**5.6.2.14 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**

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

**Returns**

The union variable.

**5.6.2.15 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**

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

**Returns**

The union variable.

**5.6.2.16 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**

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

**Returns**

The union variable.

**5.6.2.17 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**

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

**Returns**

The union variable.

**5.6.2.18 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**

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

**Returns**

The union variable.

**5.6.2.19 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**

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

**Returns**

The union variable.

**5.6.2.20 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**

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

**Returns**

The union variable.

**5.6.2.21 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**

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

**Returns**

The union variable.

**5.6.2.22 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**

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

**Returns**

The union variable.

**5.6.2.23 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**

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

**Returns**

The union variable.

**5.6.2.24 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**

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

**Returns**

The union variable.

**5.6.2.25 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**

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

**Returns**

The union variable.

**5.6.2.26 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**

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

**Returns**

The union variable.

**5.6.2.27 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**

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

**Returns**

The union variable.

**5.6.2.28 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

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

## Returns

The union variable.

### 5.6.3 Function Documentation

#### 5.6.3.1 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

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

## Returns

The union variable.

#### 5.6.3.2 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**

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

**Returns**

The union variable.

**5.6.3.3 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**

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

**Returns**

The union variable.

**5.6.3.4 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**

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

**Returns**

The union variable.

**5.6.3.5 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**

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

**Returns**

The union variable.

**5.6.3.6 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**

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

**Returns**

The union variable.

**5.6.3.7 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**

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

**Returns**

The union variable.

**5.6.3.8 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

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

## Returns

The union variable.

### 5.6.3.9 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

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

## Returns

The union variable.

### 5.6.3.10 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**

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

**Returns**

The union variable.

**5.6.3.11 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**

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

**Returns**

The union variable.

**5.6.3.12 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

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

## Returns

The union variable.

**5.6.3.13 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

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

## Returns

The union variable.

**5.6.3.14 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**

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

**Returns**

The union variable.

**5.6.3.15 gcu\_type64\_f32()**

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

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

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**

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

**Returns**

The union variable.

**5.6.3.16 gcu\_type64\_f64()**

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

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

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**

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

**Returns**

The union variable.

**5.6.3.17 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**

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

**Returns**

The union variable.

**5.6.3.18 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**

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

**Returns**

The union variable.

**5.6.3.19 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**

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

**Returns**

The union variable.

**5.6.3.20 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

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

## Returns

The union variable.

### 5.6.3.21 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

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

## Returns

The union variable.

### 5.6.3.22 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**

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

**Returns**

The union variable.

**5.6.3.23 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**

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

**Returns**

The union variable.

**5.6.3.24 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**

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

**Returns**

The union variable.

**5.6.3.25 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**

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

**Returns**

The union variable.

**5.6.3.26 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**

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

**Returns**

The union variable.

**5.6.3.27 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**

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

**Returns**

The union variable.

**5.6.3.28 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

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

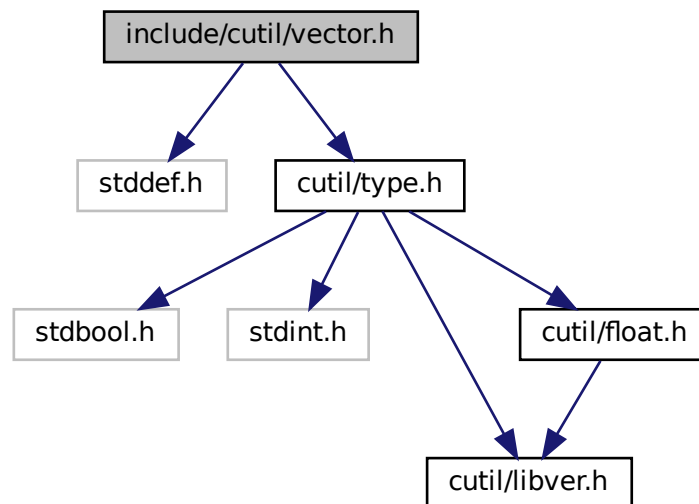
## Returns

The union variable.

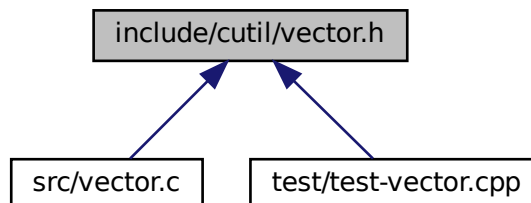
## 5.7 include/cutil/vector.h File Reference

A simple vector implementation.

```
#include <stddef.h>
#include "cutil/type.h"
Include dependency graph for vector.h:
```



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



## Classes

- struct [GCU\\_Vector64](#)  
*Container holding the information of the 64-bit vector.*
- struct [GCU\\_Vector32](#)  
*Container holding the information of the 32-bit vector.*
- struct [GCU\\_Vector16](#)  
*Container holding the information of the 16-bit vector.*
- struct [GCU\\_Vector8](#)  
*Container holding the information of the 8-bit vector.*

## Functions

- [GCU\\_Vector64](#) \* [gcu\\_vector64\\_create](#) (size\_t count)  
*Create a vector structure.*
- void [gcu\\_vector64\\_destroy](#) ([GCU\\_Vector64](#) \*vector)  
*Destroy a vector structure and clean up memory allocations.*
- bool [gcu\\_vector64\\_append](#) ([GCU\\_Vector64](#) \*vector, [GCU\\_Type64\\_Union](#) value)  
*Append an item at the end of the vector.*
- size\_t [gcu\\_vector64\\_count](#) ([GCU\\_Vector64](#) \*vector)  
*Get a count of entries in the vector.*
- [GCU\\_Vector32](#) \* [gcu\\_vector32\\_create](#) (size\_t count)  
*Create a vector structure.*
- void [gcu\\_vector32\\_destroy](#) ([GCU\\_Vector32](#) \*vector)  
*Destroy a vector structure and clean up memory allocations.*
- bool [gcu\\_vector32\\_append](#) ([GCU\\_Vector32](#) \*vector, [GCU\\_Type32\\_Union](#) value)  
*Append an item at the end of the vector.*
- size\_t [gcu\\_vector32\\_count](#) ([GCU\\_Vector32](#) \*vector)  
*Get a count of entries in the vector.*
- [GCU\\_Vector16](#) \* [gcu\\_vector16\\_create](#) (size\_t count)  
*Create a vector structure.*
- void [gcu\\_vector16\\_destroy](#) ([GCU\\_Vector16](#) \*vector)  
*Destroy a vector structure and clean up memory allocations.*
- bool [gcu\\_vector16\\_append](#) ([GCU\\_Vector16](#) \*vector, [GCU\\_Type16\\_Union](#) value)  
*Append an item at the end of the vector.*
- size\_t [gcu\\_vector16\\_count](#) ([GCU\\_Vector16](#) \*vector)  
*Get a count of entries in the vector.*
- [GCU\\_Vector8](#) \* [gcu\\_vector8\\_create](#) (size\_t count)  
*Create a vector structure.*
- void [gcu\\_vector8\\_destroy](#) ([GCU\\_Vector8](#) \*vector)  
*Destroy a vector structure and clean up memory allocations.*
- bool [gcu\\_vector8\\_append](#) ([GCU\\_Vector8](#) \*vector, [GCU\\_Type8\\_Union](#) value)  
*Append an item at the end of the vector.*
- size\_t [gcu\\_vector8\\_count](#) ([GCU\\_Vector8](#) \*vector)  
*Get a count of entries in the vector.*

### 5.7.1 Detailed Description

A simple vector implementation.

## 5.7.2 Function Documentation

### 5.7.2.1 gcu\_vector16\_append()

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

Append an item at the end of the vector.

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

#### Parameters

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

#### Returns

*true* on success, *false* otherwise.

### 5.7.2.2 gcu\_vector16\_count()

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

Get a count of entries in the vector.

#### Parameters

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

#### Returns

The count of entries in the vector.

### 5.7.2.3 gcu\_vector16\_create()

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

Create a vector structure.

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

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

#### Parameters

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

#### Returns

A struct containing the vector information.

### 5.7.2.4 `gcu_vector16_destroy()`

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

Destroy a vector structure and clean up memory allocations.

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

#### Parameters

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

### 5.7.2.5 `gcu_vector32_append()`

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

Append an item at the end of the vector.

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

#### Parameters

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



**Returns**

`true` on success, `false` otherwise.

**5.7.2.6 gcu\_vector32\_count()**

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

Get a count of entries in the vector.

**Parameters**

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

**Returns**

The count of entries in the vector.

**5.7.2.7 gcu\_vector32\_create()**

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

Create a vector structure.

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

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

**Parameters**

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

**Returns**

A struct containing the vector information.

### 5.7.2.8 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

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

### 5.7.2.9 gcu\_vector64\_append()

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

Append an item at the end of the vector.

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

#### Parameters

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

#### Returns

`true` on success, `false` otherwise.

### 5.7.2.10 gcu\_vector64\_count()

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

Get a count of entries in the vector.

#### Parameters

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

### Returns

The count of entries in the vector.

#### 5.7.2.11 gcu\_vector64\_create()

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

Create a vector structure.

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

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

### Parameters

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

### Returns

A struct containing the vector information.

#### 5.7.2.12 gcu\_vector64\_destroy()

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

Destroy a vector structure and clean up memory allocations.

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

### Parameters

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

#### 5.7.2.13 gcu\_vector8\_append()

```
bool gcu_vector8_append (
```

```
GCU_Vector8 * vector,
GCU_Type8_Union value )
```

Append an item at the end of the vector.

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

#### Parameters

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

#### Returns

true on success, false otherwise.

### 5.7.2.14 gcu\_vector8\_count()

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

Get a count of entries in the vector.

#### Parameters

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

#### Returns

The count of entries in the vector.

### 5.7.2.15 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

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

## Returns

A struct containing the vector information.

### 5.7.2.16 gcu\_vector8\_destroy()

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

Destroy a vector structure and clean up memory allocations.

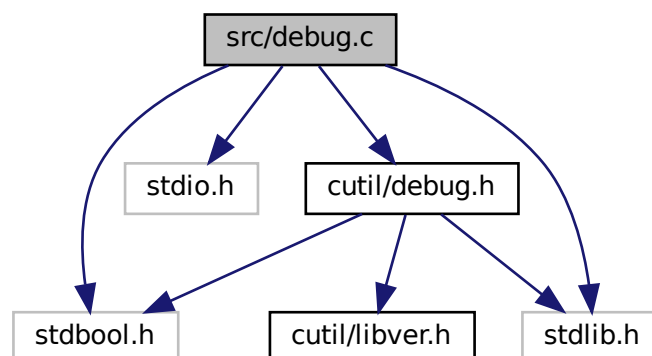
This function will not address any memory allocations of the elements themselves (if any). The programmer is responsible for controlling any memory management on behalf of the elements.

## Parameters

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

## 5.8 src/debug.c File Reference

```
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include "cutil/debug.h"
Include dependency graph for debug.c:
```



## Functions

- void \* [gcu\\_malloc](#) (size\_t size, const char \*file, size\_t line)  
*Wrapper for the standard malloc() function.*
- void \* [gcu\\_calloc](#) (size\_t nitems, size\_t size, const char \*file, size\_t line)  
*Wrapper for the standard calloc() function.*
- void \* [gcu\\_realloc](#) (void \*pointer, size\_t size, const char \*file, size\_t line)  
*Wrapper for the standard realloc() function.*
- void [gcu\\_free](#) (void \*pointer, const char \*file, size\_t line)  
*Wrapper for the standard free() function.*
- void [gcu\\_mem\\_start](#) (void)  
*Signal that intercepted memory management calls should be logged to stderr.*
- void [gcu\\_mem\\_stop](#) (void)  
*Signal that intercepted memory management calls should no longer be logged to stderr.*

## Variables

- bool **capture** = true

### 5.8.1 Function Documentation

#### 5.8.1.1 gcu\_calloc()

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

Wrapper for the standard calloc() function.

##### Parameters

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

##### Returns

The beginning byte of the allocated memory.

### 5.8.1.2 gcu\_free()

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

Wrapper for the standard free() function.

#### Parameters

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

### 5.8.1.3 gcu\_malloc()

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

Wrapper for the standard malloc() function.

#### Parameters

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

#### Returns

The beginning byte of the allocated memory.

### 5.8.1.4 gcu\_realloc()

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

Wrapper for the standard realloc() function.

**Parameters**

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

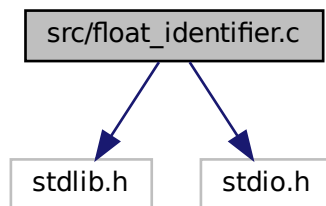
**Returns**

The beginning byte of the reallocated memory.

## 5.9 src/float\_identifier.c File Reference

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

```
#include <stdlib.h>
#include <stdio.h>
Include dependency graph for float_identifier.c:
```

**Functions**

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

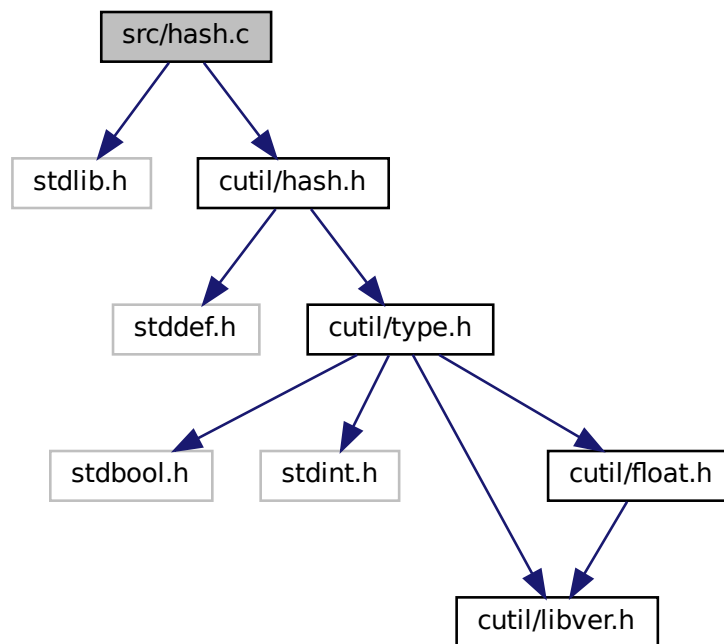
### 5.9.1 Detailed Description

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



## 5.10 src/hash.c File Reference

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



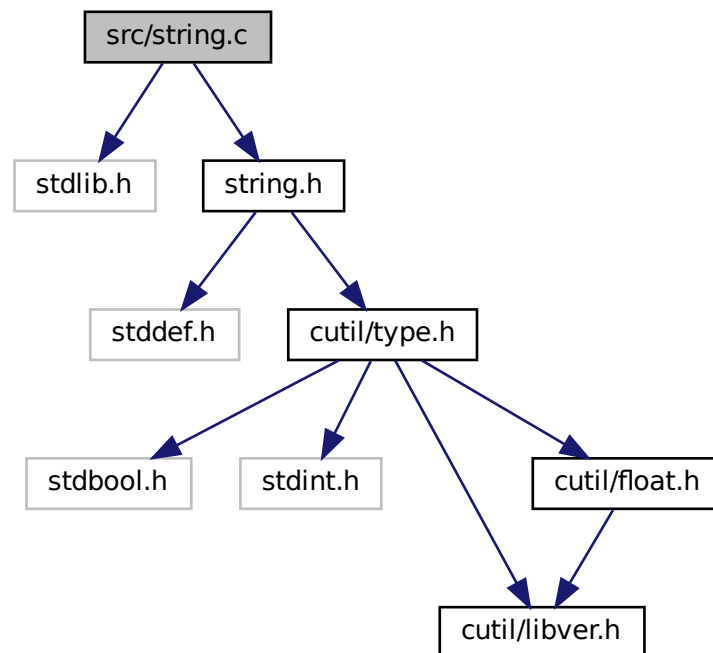
### Macros

- `#define GROWTH_FACTOR 1.25`
- `#define BITDEPTH 64`
- `#define DEFAULT_TYPE gcu\_type64\_ui64`
- `#define BITDEPTH 32`
- `#define DEFAULT_TYPE gcu\_type32\_ui32`
- `#define BITDEPTH 16`
- `#define DEFAULT_TYPE gcu\_type16\_ui16`
- `#define BITDEPTH 8`
- `#define DEFAULT_TYPE gcu\_type8\_ui8`

## 5.11 src/string.c File Reference

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

Include dependency graph for string.c:



## Macros

- `#define ROTL32(a, b) ((a << b) | (a >> (32 - b)))`
- `#define ROTL64(a, b) ((a << b) | (a >> (64 - b)))`

## 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.
- `size_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.

### 5.11.1 Function Documentation

### 5.11.1.1 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

|            |  |
|------------|--|
| <i>str</i> | A pointer to the string (or data block). |
| <i>len</i> | The length of the data in bytes.         |

#### Returns

A 32-bit number representing the value.

Here is the call graph for this function:



### 5.11.1.2 gcu\_string\_hash\_64()

```
size_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.

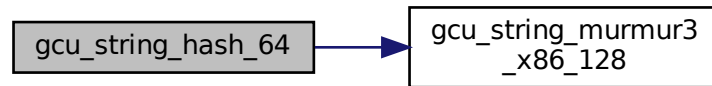
#### Parameters

|            |  |
|------------|--|
| <i>str</i> | A pointer to the string (or data block). |
| <i>len</i> | The length of the data in bytes.         |

**Returns**

A 64-bit number representing the value.

Here is the call graph for this function:

**5.11.1.3 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**

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

**5.11.1.4 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**

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

**5.11.1.5 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>

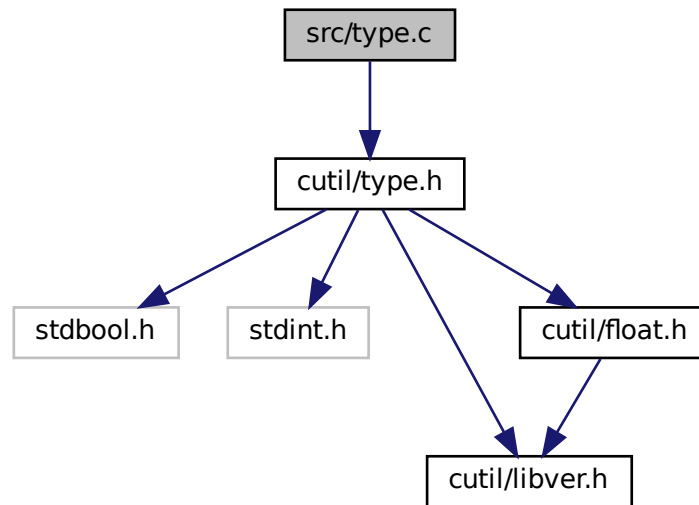
**Parameters**

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

**5.12 src/type.c File Reference**

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

Include dependency graph for type.c:



## Functions

- [GCU\\_Type64\\_Union gcu\\_type64\\_p](#) (void \*val)  
Create a 64-bit union variable with the type `void *`.
- [GCU\\_Type64\\_Union gcu\\_type64\\_ui64](#) (uint64\_t val)  
Create a 64-bit union variable with the type `uint64_t`.
- [GCU\\_Type64\\_Union gcu\\_type64\\_ui32](#) (uint32\_t val)  
Create a 64-bit union variable with the type `uint32_t`.
- [GCU\\_Type64\\_Union gcu\\_type64\\_ui16](#) (uint16\_t val)  
Create a 64-bit union variable with the type `uint16_t`.
- [GCU\\_Type64\\_Union gcu\\_type64\\_ui8](#) (uint8\_t val)  
Create a 64-bit union variable with the type `uint8_t`.
- [GCU\\_Type64\\_Union gcu\\_type64\\_i64](#) (int64\_t val)  
Create a 64-bit union variable with the type `int64_t`.
- [GCU\\_Type64\\_Union gcu\\_type64\\_i32](#) (int32\_t val)  
Create a 64-bit union variable with the type `int32_t`.
- [GCU\\_Type64\\_Union gcu\\_type64\\_i16](#) (int16\_t val)  
Create a 64-bit union variable with the type `int16_t`.
- [GCU\\_Type64\\_Union gcu\\_type64\\_i8](#) (int8\_t val)  
Create a 64-bit union variable with the type `int8_t`.
- [GCU\\_Type64\\_Union gcu\\_type64\\_f64](#) (GCU\_float64\_t val)  
Create a 64-bit union variable with the type `float` with 64 bits.
- [GCU\\_Type64\\_Union gcu\\_type64\\_f32](#) (GCU\_float32\_t val)  
Create a 64-bit union variable with the type `float` with 32 bits.
- [GCU\\_Type64\\_Union gcu\\_type64\\_c](#) (char val)  
Create a 64-bit union variable with the type `char`.
- [GCU\\_Type32\\_Union gcu\\_type32\\_ui32](#) (uint32\_t val)

- Create a 32-bit union variable with the type `uint32_t`.*
  - [GCU\\_Type32\\_Union gcu\\_type32\\_ui16](#) (`uint16_t val`)
- Create a 32-bit union variable with the type `uint16_t`.*
  - [GCU\\_Type32\\_Union gcu\\_type32\\_ui8](#) (`uint8_t val`)
- Create a 32-bit union variable with the type `uint8_t`.*
  - [GCU\\_Type32\\_Union gcu\\_type32\\_i32](#) (`int32_t val`)
- Create a 32-bit union variable with the type `int32_t`.*
  - [GCU\\_Type32\\_Union gcu\\_type32\\_i16](#) (`int16_t val`)
- Create a 32-bit union variable with the type `int16_t`.*
  - [GCU\\_Type32\\_Union gcu\\_type32\\_i8](#) (`int8_t val`)
- Create a 32-bit union variable with the type `int8_t`.*
  - [GCU\\_Type32\\_Union gcu\\_type32\\_f32](#) (`GCU_float32_t val`)
- Create a 32-bit union variable with the type `float` with 32 bits.*
  - [GCU\\_Type32\\_Union gcu\\_type32\\_c](#) (`char val`)
- Create a 32-bit union variable with the type `char`.*
  - [GCU\\_Type16\\_Union gcu\\_type16\\_ui16](#) (`uint16_t val`)
- Create a 16-bit union variable with the type `uint16_t`.*
  - [GCU\\_Type16\\_Union gcu\\_type16\\_ui8](#) (`uint8_t val`)
- Create a 16-bit union variable with the type `uint8_t`.*
  - [GCU\\_Type16\\_Union gcu\\_type16\\_i16](#) (`int16_t val`)
- Create a 16-bit union variable with the type `int16_t`.*
  - [GCU\\_Type16\\_Union gcu\\_type16\\_i8](#) (`int8_t val`)
- Create a 16-bit union variable with the type `int8_t`.*
  - [GCU\\_Type16\\_Union gcu\\_type16\\_c](#) (`char val`)
- Create a 16-bit union variable with the type `char`.*
  - [GCU\\_Type8\\_Union gcu\\_type8\\_ui8](#) (`uint8_t val`)
- Create a 8-bit union variable with the type `uint8_t`.*
  - [GCU\\_Type8\\_Union gcu\\_type8\\_i8](#) (`int8_t val`)
- Create a 8-bit union variable with the type `int8_t`.*
  - [GCU\\_Type8\\_Union gcu\\_type8\\_c](#) (`char val`)
- Create a 8-bit union variable with the type `char`.*

## 5.12.1 Function Documentation

### 5.12.1.1 `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**

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

**Returns**

The union variable.

**5.12.1.2 gcu\_type16\_i16()**

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

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

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**

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

**Returns**

The union variable.

**5.12.1.3 gcu\_type16\_i8()**

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

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

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**

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

**Returns**

The union variable.

**5.12.1.4 gcu\_type16\_ui16()**

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

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

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**

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

**Returns**

The union variable.

**5.12.1.5 gcu\_type16\_ui8()**

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

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

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**

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

**Returns**

The union variable.

**5.12.1.6 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**

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

**Returns**

The union variable.

**5.12.1.7 gcu\_type32\_f32()**

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

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

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**

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

**Returns**

The union variable.

**5.12.1.8 gcu\_type32\_i16()**

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

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

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**

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

**Returns**

The union variable.

**5.12.1.9 gcu\_type32\_i32()**

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

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

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

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

## Returns

The union variable.

**5.12.1.10 gcu\_type32\_i8()**

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

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

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

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

## Returns

The union variable.

**5.12.1.11 gcu\_type32\_ui16()**

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

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

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**

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

**Returns**

The union variable.

**5.12.1.12 gcu\_type32\_ui32()**

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

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

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**

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

**Returns**

The union variable.

**5.12.1.13 gcu\_type32\_ui8()**

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

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

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

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

## Returns

The union variable.

**5.12.1.14 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

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

## Returns

The union variable.

**5.12.1.15 gcu\_type64\_f32()**

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

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

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**

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

**Returns**

The union variable.

**5.12.1.16 gcu\_type64\_f64()**

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

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

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**

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

**Returns**

The union variable.

**5.12.1.17 gcu\_type64\_i16()**

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

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

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**

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

**Returns**

The union variable.

**5.12.1.18 gcu\_type64\_i32()**

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

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

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**

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

**Returns**

The union variable.

**5.12.1.19 gcu\_type64\_i64()**

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

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

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**

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

**Returns**

The union variable.

**5.12.1.20 gcu\_type64\_i8()**

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

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

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**

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

**Returns**

The union variable.

**5.12.1.21 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

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

## Returns

The union variable.

**5.12.1.22 gcu\_type64\_ui16()**

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

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

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

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

## Returns

The union variable.

**5.12.1.23 gcu\_type64\_ui32()**

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

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

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**

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

**Returns**

The union variable.

**5.12.1.24 gcu\_type64\_ui64()**

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

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

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**

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

**Returns**

The union variable.

**5.12.1.25 gcu\_type64\_ui8()**

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

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

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**

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

**Returns**

The union variable.

**5.12.1.26 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**

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

**Returns**

The union variable.

**5.12.1.27 gcu\_type8\_i8()**

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

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

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**

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

**Returns**

The union variable.

**5.12.1.28 gcu\_type8\_ui8()**

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

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

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**

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

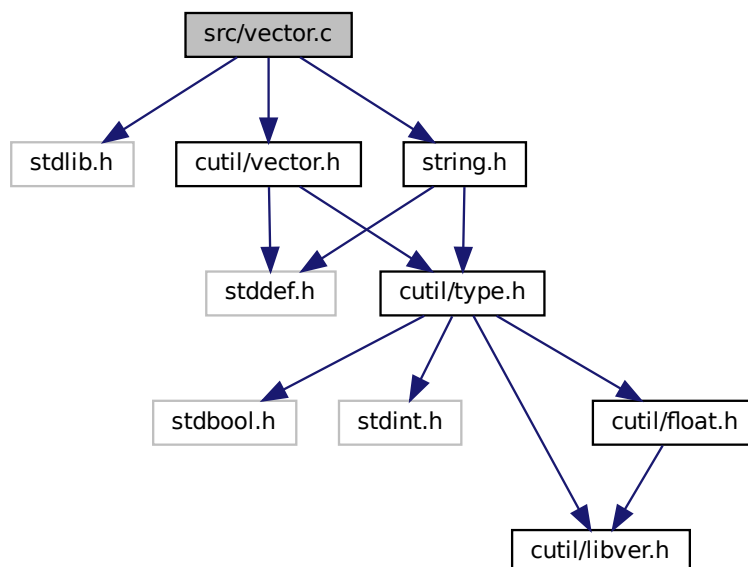
**Returns**

The union variable.

**5.13 src/vector.c File Reference**

```
#include <stdlib.h>  
#include <string.h>  
#include "cutil/vector.h"  
#include "vector.template.c"
```

Include dependency graph for vector.c:



## Macros

- `#define GROWTH_FACTOR 1.3`
- `#define BITDEPTH 64`
- `#define BITDEPTH 32`
- `#define BITDEPTH 16`
- `#define BITDEPTH 8`

## 5.14 test/test-debug.cpp File Reference

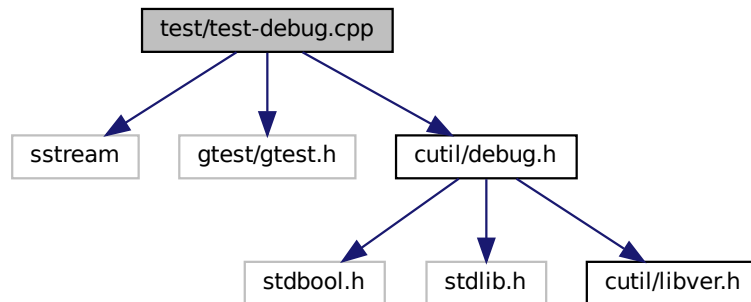
Test the behavior of Ghoti.io CUtil debug library and tools.

```

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

```

Include dependency graph for test-debug.cpp:



## Functions

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

### 5.14.1 Detailed Description

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

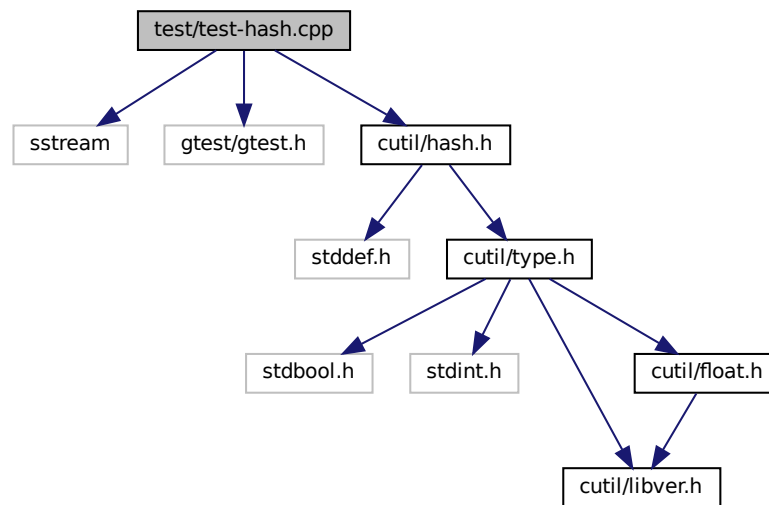
## 5.15 test/test-hash.cpp File Reference

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

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



Include dependency graph for test-hash.cpp:



## Functions

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

### 5.15.1 Detailed Description

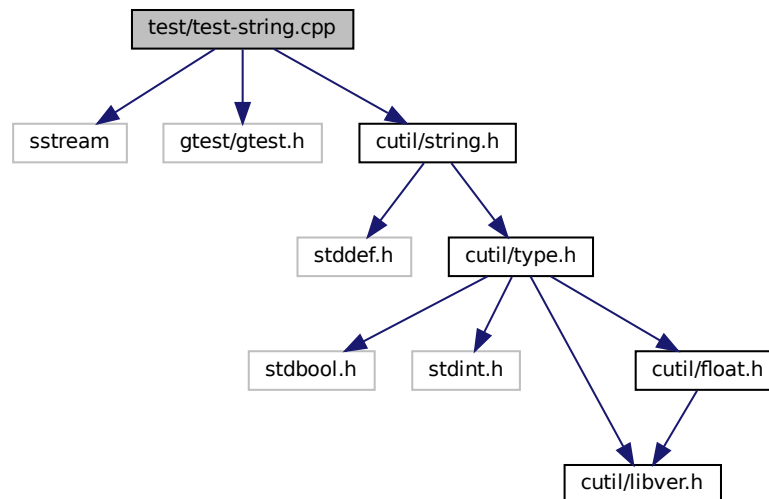
Test the behavior of Ghoti.io CUtil hash table library.

## 5.16 test/test-string.cpp File Reference

Test the behavior of Ghoti.io CUtil string library.

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

Include dependency graph for test-string.cpp:



### Functions

- **TEST** (Murmur3, SeedCheck)
- **TEST** (Murmur3, HashCheck)
- **TEST** (Murmur3, Length)
- **TEST** (Hash, HelperFunction32)
- **TEST** (Hash, HelperFunction64)
- int **main** (int argc, char \*\*argv)

### 5.16.1 Detailed Description

Test the behavior of Ghoti.io CUtil string library.

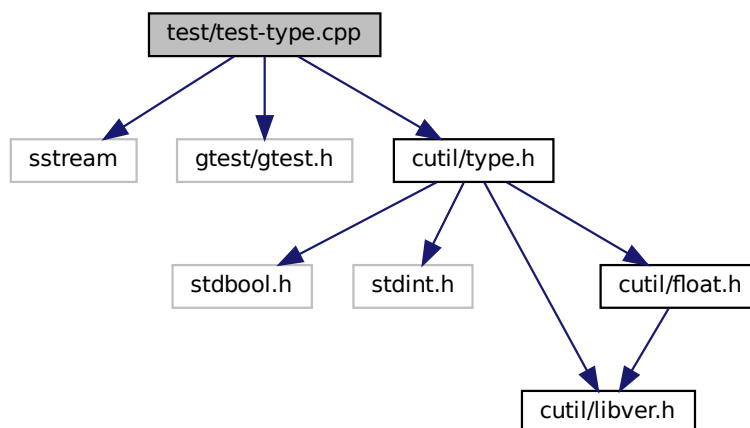
## 5.17 test/test-type.cpp File Reference

Test the behavior of Ghoti.io CUtil type library.

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

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

Include dependency graph for test-type.cpp:



## Functions

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

### 5.17.1 Detailed Description

Test the behavior of Ghoti.io CUtil type library.

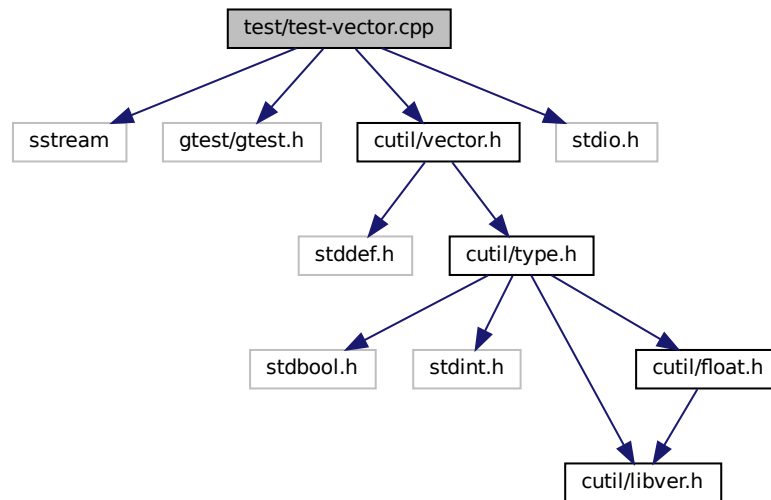
## 5.18 test/test-vector.cpp File Reference

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

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

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

Include dependency graph for test-vector.cpp:



## Functions

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

### 5.18.1 Detailed Description

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

# Index

- debug.c
  - gcu\_calloc, [100](#)
  - gcu\_free, [100](#)
  - gcu\_malloc, [101](#)
  - gcu\_realloc, [101](#)
- debug.h
  - gcu\_calloc, [30](#)
  - gcu\_free, [31](#)
  - gcu\_malloc, [31](#)
  - gcu\_realloc, [31](#)
- gcu\_calloc
  - debug.c, [100](#)
  - debug.h, [30](#)
- gcu\_free
  - debug.c, [100](#)
  - debug.h, [31](#)
- GCU\_Hash16\_Cell, [7](#)
- gcu\_hash16\_contains
  - hash.h, [36](#)
- gcu\_hash16\_count
  - hash.h, [36](#)
- gcu\_hash16\_create
  - hash.h, [37](#)
- gcu\_hash16\_destroy
  - hash.h, [37](#)
- gcu\_hash16\_get
  - hash.h, [37](#)
- GCU\_Hash16\_Iterator, [8](#)
- gcu\_hash16\_iterator\_get
  - hash.h, [38](#)
- gcu\_hash16\_iterator\_next
  - hash.h, [38](#)
- gcu\_hash16\_remove
  - hash.h, [39](#)
- gcu\_hash16\_set
  - hash.h, [39](#)
- GCU\_Hash16\_Table, [9](#)
- GCU\_Hash16\_Value, [10](#)
- GCU\_Hash32\_Cell, [11](#)
- gcu\_hash32\_contains
  - hash.h, [39](#)
- gcu\_hash32\_count
  - hash.h, [40](#)
- gcu\_hash32\_create
  - hash.h, [40](#)
- gcu\_hash32\_destroy
  - hash.h, [41](#)
- gcu\_hash32\_get
  - hash.h, [41](#)
- GCU\_Hash32\_Iterator, [12](#)
- gcu\_hash32\_iterator\_get
  - hash.h, [41](#)
- gcu\_hash32\_iterator\_next
  - hash.h, [43](#)
- gcu\_hash32\_remove
  - hash.h, [43](#)
- gcu\_hash32\_set
  - hash.h, [44](#)
- GCU\_Hash32\_Table, [13](#)
- GCU\_Hash32\_Value, [14](#)
- GCU\_Hash64\_Cell, [15](#)
- gcu\_hash64\_contains
  - hash.h, [44](#)
- gcu\_hash64\_count
  - hash.h, [44](#)
- gcu\_hash64\_create
  - hash.h, [45](#)
- gcu\_hash64\_destroy
  - hash.h, [45](#)
- gcu\_hash64\_get
  - hash.h, [46](#)
- GCU\_Hash64\_Iterator, [16](#)
- gcu\_hash64\_iterator\_get
  - hash.h, [46](#)
- gcu\_hash64\_iterator\_next
  - hash.h, [46](#)
- gcu\_hash64\_remove
  - hash.h, [47](#)
- gcu\_hash64\_set
  - hash.h, [47](#)
- GCU\_Hash64\_Table, [17](#)
- GCU\_Hash64\_Value, [18](#)
- GCU\_Hash8\_Cell, [19](#)
- gcu\_hash8\_contains
  - hash.h, [48](#)
- gcu\_hash8\_count
  - hash.h, [48](#)
- gcu\_hash8\_create
  - hash.h, [48](#)
- gcu\_hash8\_destroy
  - hash.h, [49](#)
- gcu\_hash8\_get
  - hash.h, [49](#)
- GCU\_Hash8\_Iterator, [20](#)
- gcu\_hash8\_iterator\_get
  - hash.h, [50](#)
- gcu\_hash8\_iterator\_next
  - hash.h, [50](#)

- gcu\_hash8\_remove
  - hash.h, [50](#)
- gcu\_hash8\_set
  - hash.h, [51](#)
- GCU\_Hash8\_Table, [21](#)
- GCU\_Hash8\_Value, [22](#)
- gcu\_malloc
  - debug.c, [101](#)
  - debug.h, [31](#)
- gcu\_realloc
  - debug.c, [101](#)
  - debug.h, [31](#)
- gcu\_string\_hash\_32
  - string.c, [104](#)
  - string.h, [57](#)
- gcu\_string\_hash\_64
  - string.c, [105](#)
  - string.h, [57](#)
- gcu\_string\_murmur3\_32
  - string.c, [106](#)
  - string.h, [58](#)
- gcu\_string\_murmur3\_x64\_128
  - string.c, [106](#)
  - string.h, [59](#)
- gcu\_string\_murmur3\_x86\_128
  - string.c, [108](#)
  - string.h, [59](#)
- GCU\_TYPE16\_C
  - type.h, [63](#)
- gcu\_type16\_c
  - type.c, [110](#)
  - type.h, [77](#)
- GCU\_TYPE16\_I16
  - type.h, [63](#)
- gcu\_type16\_i16
  - type.c, [111](#)
  - type.h, [77](#)
- GCU\_TYPE16\_I8
  - type.h, [64](#)
- gcu\_type16\_i8
  - type.c, [111](#)
  - type.h, [78](#)
- GCU\_TYPE16\_UI16
  - type.h, [64](#)
- gcu\_type16\_ui16
  - type.c, [112](#)
  - type.h, [78](#)
- GCU\_TYPE16\_UI8
  - type.h, [65](#)
- gcu\_type16\_ui8
  - type.c, [112](#)
  - type.h, [79](#)
- GCU\_Type16\_Union, [23](#)
- GCU\_TYPE32\_C
  - type.h, [65](#)
- gcu\_type32\_c
  - type.c, [113](#)
  - type.h, [79](#)
- GCU\_TYPE32\_F32
  - type.h, [66](#)
- gcu\_type32\_f32
  - type.c, [113](#)
  - type.h, [80](#)
- GCU\_TYPE32\_I16
  - type.h, [66](#)
- gcu\_type32\_i16
  - type.c, [114](#)
  - type.h, [80](#)
- GCU\_TYPE32\_I32
  - type.h, [67](#)
- gcu\_type32\_i32
  - type.c, [114](#)
  - type.h, [81](#)
- GCU\_TYPE32\_I8
  - type.h, [67](#)
- gcu\_type32\_i8
  - type.c, [115](#)
  - type.h, [81](#)
- GCU\_TYPE32\_UI16
  - type.h, [68](#)
- gcu\_type32\_ui16
  - type.c, [115](#)
  - type.h, [82](#)
- GCU\_TYPE32\_UI32
  - type.h, [68](#)
- gcu\_type32\_ui32
  - type.c, [116](#)
  - type.h, [82](#)
- GCU\_TYPE32\_UI8
  - type.h, [69](#)
- gcu\_type32\_ui8
  - type.c, [116](#)
  - type.h, [83](#)
- GCU\_Type32\_Union, [23](#)
- GCU\_TYPE64\_C
  - type.h, [69](#)
- gcu\_type64\_c
  - type.c, [117](#)
  - type.h, [83](#)
- GCU\_TYPE64\_F32
  - type.h, [70](#)
- gcu\_type64\_f32
  - type.c, [117](#)
  - type.h, [84](#)
- GCU\_TYPE64\_F64
  - type.h, [70](#)
- gcu\_type64\_f64
  - type.c, [118](#)
  - type.h, [84](#)
- GCU\_TYPE64\_I16
  - type.h, [71](#)
- gcu\_type64\_i16
  - type.c, [118](#)
  - type.h, [85](#)
- GCU\_TYPE64\_I32
  - type.h, [71](#)

gcu\_type64\_i32  
  type.c, [119](#)  
  type.h, [85](#)  
GCU\_TYPE64\_I64  
  type.h, [72](#)  
gcu\_type64\_i64  
  type.c, [119](#)  
  type.h, [86](#)  
GCU\_TYPE64\_I8  
  type.h, [72](#)  
gcu\_type64\_i8  
  type.c, [120](#)  
  type.h, [86](#)  
GCU\_TYPE64\_P  
  type.h, [73](#)  
gcu\_type64\_p  
  type.c, [120](#)  
  type.h, [87](#)  
GCU\_TYPE64\_UI16  
  type.h, [73](#)  
gcu\_type64\_ui16  
  type.c, [121](#)  
  type.h, [87](#)  
GCU\_TYPE64\_UI32  
  type.h, [74](#)  
gcu\_type64\_ui32  
  type.c, [121](#)  
  type.h, [88](#)  
GCU\_TYPE64\_UI64  
  type.h, [74](#)  
gcu\_type64\_ui64  
  type.c, [122](#)  
  type.h, [88](#)  
GCU\_TYPE64\_UI8  
  type.h, [75](#)  
gcu\_type64\_ui8  
  type.c, [122](#)  
  type.h, [89](#)  
GCU\_Type64\_Union, [24](#)  
GCU\_TYPE8\_C  
  type.h, [75](#)  
gcu\_type8\_c  
  type.c, [123](#)  
  type.h, [89](#)  
GCU\_TYPE8\_I8  
  type.h, [76](#)  
gcu\_type8\_i8  
  type.c, [123](#)  
  type.h, [90](#)  
GCU\_TYPE8\_UI8  
  type.h, [76](#)  
gcu\_type8\_ui8  
  type.c, [124](#)  
  type.h, [90](#)  
GCU\_Type8\_Union, [24](#)  
GCU\_Vector16, [25](#)  
gcu\_vector16\_append  
  vector.h, [93](#)  
gcu\_vector16\_count  
  vector.h, [93](#)  
gcu\_vector16\_create  
  vector.h, [93](#)  
gcu\_vector16\_destroy  
  vector.h, [94](#)  
GCU\_Vector32, [26](#)  
gcu\_vector32\_append  
  vector.h, [94](#)  
gcu\_vector32\_count  
  vector.h, [95](#)  
gcu\_vector32\_create  
  vector.h, [95](#)  
gcu\_vector32\_destroy  
  vector.h, [95](#)  
GCU\_Vector64, [27](#)  
gcu\_vector64\_append  
  vector.h, [96](#)  
gcu\_vector64\_count  
  vector.h, [96](#)  
gcu\_vector64\_create  
  vector.h, [97](#)  
gcu\_vector64\_destroy  
  vector.h, [97](#)  
GCU\_Vector8, [28](#)  
gcu\_vector8\_append  
  vector.h, [97](#)  
gcu\_vector8\_count  
  vector.h, [98](#)  
gcu\_vector8\_create  
  vector.h, [98](#)  
gcu\_vector8\_destroy  
  vector.h, [99](#)  
GHOTIIO\_CUTIL  
  libver.h, [52](#)  
GHOTIIO\_CUTIL\_CONCAT2  
  libver.h, [53](#)  
GHOTIIO\_CUTIL\_CONCAT2\_INNER  
  libver.h, [53](#)  
GHOTIIO\_CUTIL\_CONCAT3  
  libver.h, [54](#)  
GHOTIIO\_CUTIL\_CONCAT3\_INNER  
  libver.h, [54](#)  
GHOTIIO\_CUTIL\_NAME  
  libver.h, [54](#)  
GHOTIIO\_CUTIL\_RENAME  
  libver.h, [55](#)  
GHOTIIO\_CUTIL\_RENAME\_INNER  
  libver.h, [55](#)  
hash.h  
  gcu\_hash16\_contains, [36](#)  
  gcu\_hash16\_count, [36](#)  
  gcu\_hash16\_create, [37](#)  
  gcu\_hash16\_destroy, [37](#)  
  gcu\_hash16\_get, [37](#)  
  gcu\_hash16\_iterator\_get, [38](#)  
  gcu\_hash16\_iterator\_next, [38](#)  
  gcu\_hash16\_remove, [39](#)

- gcu\_hash16\_set, [39](#)
- gcu\_hash32\_contains, [39](#)
- gcu\_hash32\_count, [40](#)
- gcu\_hash32\_create, [40](#)
- gcu\_hash32\_destroy, [41](#)
- gcu\_hash32\_get, [41](#)
- gcu\_hash32\_iterator\_get, [41](#)
- gcu\_hash32\_iterator\_next, [43](#)
- gcu\_hash32\_remove, [43](#)
- gcu\_hash32\_set, [44](#)
- gcu\_hash64\_contains, [44](#)
- gcu\_hash64\_count, [44](#)
- gcu\_hash64\_create, [45](#)
- gcu\_hash64\_destroy, [45](#)
- gcu\_hash64\_get, [46](#)
- gcu\_hash64\_iterator\_get, [46](#)
- gcu\_hash64\_iterator\_next, [46](#)
- gcu\_hash64\_remove, [47](#)
- gcu\_hash64\_set, [47](#)
- gcu\_hash8\_contains, [48](#)
- gcu\_hash8\_count, [48](#)
- gcu\_hash8\_create, [48](#)
- gcu\_hash8\_destroy, [49](#)
- gcu\_hash8\_get, [49](#)
- gcu\_hash8\_iterator\_get, [50](#)
- gcu\_hash8\_iterator\_next, [50](#)
- gcu\_hash8\_remove, [50](#)
- gcu\_hash8\_set, [51](#)
- include/cutil/debug.h, [29](#)
- include/cutil/float.h, [32](#)
- include/cutil/hash.h, [33](#)
- include/cutil/libver.h, [51](#)
- include/cutil/string.h, [56](#)
- include/cutil/type.h, [60](#)
- include/cutil/vector.h, [91](#)
- libver.h
  - GHOTIO\_CUTIL, [52](#)
  - GHOTIO\_CUTIL\_CONCAT2, [53](#)
  - GHOTIO\_CUTIL\_CONCAT2\_INNER, [53](#)
  - GHOTIO\_CUTIL\_CONCAT3, [54](#)
  - GHOTIO\_CUTIL\_CONCAT3\_INNER, [54](#)
  - GHOTIO\_CUTIL\_NAME, [54](#)
  - GHOTIO\_CUTIL\_RENAME, [55](#)
  - GHOTIO\_CUTIL\_RENAME\_INNER, [55](#)
- src/debug.c, [99](#)
- src/float\_identifier.c, [102](#)
- src/hash.c, [103](#)
- src/string.c, [103](#)
- src/type.c, [108](#)
- src/vector.c, [124](#)
- string.c
  - gcu\_string\_hash\_32, [104](#)
  - gcu\_string\_hash\_64, [105](#)
  - gcu\_string\_murmur3\_32, [106](#)
  - gcu\_string\_murmur3\_x64\_128, [106](#)
  - gcu\_string\_murmur3\_x86\_128, [108](#)
- string.h
  - gcu\_string\_hash\_32, [57](#)
  - gcu\_string\_hash\_64, [57](#)
  - gcu\_string\_murmur3\_32, [58](#)
  - gcu\_string\_murmur3\_x64\_128, [59](#)
  - gcu\_string\_murmur3\_x86\_128, [59](#)
- test/test-debug.cpp, [125](#)
- test/test-hash.cpp, [126](#)
- test/test-string.cpp, [128](#)
- test/test-type.cpp, [128](#)
- test/test-vector.cpp, [129](#)
- type.c
  - gcu\_type16\_c, [110](#)
  - gcu\_type16\_i16, [111](#)
  - gcu\_type16\_i8, [111](#)
  - gcu\_type16\_ui16, [112](#)
  - gcu\_type16\_ui8, [112](#)
  - gcu\_type32\_c, [113](#)
  - gcu\_type32\_f32, [113](#)
  - gcu\_type32\_i16, [114](#)
  - gcu\_type32\_i32, [114](#)
  - gcu\_type32\_i8, [115](#)
  - gcu\_type32\_ui16, [115](#)
  - gcu\_type32\_ui32, [116](#)
  - gcu\_type32\_ui8, [116](#)
  - gcu\_type64\_c, [117](#)
  - gcu\_type64\_f32, [117](#)
  - gcu\_type64\_f64, [118](#)
  - gcu\_type64\_i16, [118](#)
  - gcu\_type64\_i32, [119](#)
  - gcu\_type64\_i64, [119](#)
  - gcu\_type64\_i8, [120](#)
  - gcu\_type64\_p, [120](#)
  - gcu\_type64\_ui16, [121](#)
  - gcu\_type64\_ui32, [121](#)
  - gcu\_type64\_ui64, [122](#)
  - gcu\_type64\_ui8, [122](#)
  - gcu\_type8\_c, [123](#)
  - gcu\_type8\_i8, [123](#)
  - gcu\_type8\_ui8, [124](#)
- type.h
  - GCU\_TYPE16\_C, [63](#)
  - gcu\_type16\_c, [77](#)
  - GCU\_TYPE16\_I16, [63](#)
  - gcu\_type16\_i16, [77](#)
  - GCU\_TYPE16\_I8, [64](#)
  - gcu\_type16\_i8, [78](#)
  - GCU\_TYPE16\_UI16, [64](#)
  - gcu\_type16\_ui16, [78](#)
  - GCU\_TYPE16\_UI8, [65](#)
  - gcu\_type16\_ui8, [79](#)
  - GCU\_TYPE32\_C, [65](#)
  - gcu\_type32\_c, [79](#)
  - GCU\_TYPE32\_F32, [66](#)
  - gcu\_type32\_f32, [80](#)
  - GCU\_TYPE32\_I16, [66](#)
  - gcu\_type32\_i16, [80](#)
  - GCU\_TYPE32\_I32, [67](#)



- gcu\_type32\_i32, [81](#)
- GCU\_TYPE32\_I8, [67](#)
- gcu\_type32\_i8, [81](#)
- GCU\_TYPE32\_UI16, [68](#)
- gcu\_type32\_ui16, [82](#)
- GCU\_TYPE32\_UI32, [68](#)
- gcu\_type32\_ui32, [82](#)
- GCU\_TYPE32\_UI8, [69](#)
- gcu\_type32\_ui8, [83](#)
- GCU\_TYPE64\_C, [69](#)
- gcu\_type64\_c, [83](#)
- GCU\_TYPE64\_F32, [70](#)
- gcu\_type64\_f32, [84](#)
- GCU\_TYPE64\_F64, [70](#)
- gcu\_type64\_f64, [84](#)
- GCU\_TYPE64\_I16, [71](#)
- gcu\_type64\_i16, [85](#)
- GCU\_TYPE64\_I32, [71](#)
- gcu\_type64\_i32, [85](#)
- GCU\_TYPE64\_I64, [72](#)
- gcu\_type64\_i64, [86](#)
- GCU\_TYPE64\_I8, [72](#)
- gcu\_type64\_i8, [86](#)
- GCU\_TYPE64\_P, [73](#)
- gcu\_type64\_p, [87](#)
- GCU\_TYPE64\_UI16, [73](#)
- gcu\_type64\_ui16, [87](#)
- GCU\_TYPE64\_UI32, [74](#)
- gcu\_type64\_ui32, [88](#)
- GCU\_TYPE64\_UI64, [74](#)
- gcu\_type64\_ui64, [88](#)
- GCU\_TYPE64\_UI8, [75](#)
- gcu\_type64\_ui8, [89](#)
- GCU\_TYPE8\_C, [75](#)
- gcu\_type8\_c, [89](#)
- GCU\_TYPE8\_I8, [76](#)
- gcu\_type8\_i8, [90](#)
- GCU\_TYPE8\_UI8, [76](#)
- gcu\_type8\_ui8, [90](#)

#### vector.h

- gcu\_vector16\_append, [93](#)
- gcu\_vector16\_count, [93](#)
- gcu\_vector16\_create, [93](#)
- gcu\_vector16\_destroy, [94](#)
- gcu\_vector32\_append, [94](#)
- gcu\_vector32\_count, [95](#)
- gcu\_vector32\_create, [95](#)
- gcu\_vector32\_destroy, [95](#)
- gcu\_vector64\_append, [96](#)
- gcu\_vector64\_count, [96](#)
- gcu\_vector64\_create, [97](#)
- gcu\_vector64\_destroy, [97](#)
- gcu\_vector8\_append, [97](#)
- gcu\_vector8\_count, [98](#)
- gcu\_vector8\_create, [98](#)
- gcu\_vector8\_destroy, [99](#)