

libgenerics

Generated by Doxygen 1.8.12

Contents

1	Class Index	1
1.1	Class List	1
2	File Index	3
2.1	File List	3
3	Class Documentation	5
3.1	graph_t Struct Reference	5
3.1.1	Detailed Description	6
3.1.2	Member Data Documentation	6
3.1.2.1	adj	6
3.1.2.2	E	6
3.1.2.3	label	6
3.1.2.4	member_size	6
3.1.2.5	V	6
3.2	priority_queue_t Struct Reference	6
3.2.1	Member Data Documentation	7
3.2.1.1	compare	7
3.2.1.2	compare_argument	7
3.2.1.3	member_size	7
3.2.1.4	queue	7
3.2.1.5	size	7
3.3	qnode_t Struct Reference	7
3.3.1	Detailed Description	8

3.3.2	Member Data Documentation	8
3.3.2.1	data	8
3.3.2.2	next	8
3.3.2.3	prev	8
3.4	queue_t Struct Reference	8
3.4.1	Detailed Description	9
3.4.2	Member Data Documentation	9
3.4.2.1	head	9
3.4.2.2	member_size	9
3.4.2.3	size	9
3.4.2.4	tail	9
3.5	snode_t Struct Reference	9
3.5.1	Detailed Description	10
3.5.2	Member Data Documentation	10
3.5.2.1	data	10
3.5.2.2	next	10
3.5.2.3	prev	10
3.6	stack_t Struct Reference	10
3.6.1	Detailed Description	11
3.6.2	Member Data Documentation	11
3.6.2.1	head	11
3.6.2.2	member_size	11
3.6.2.3	size	11
3.7	tnode_t Struct Reference	11
3.7.1	Detailed Description	12
3.7.2	Member Data Documentation	12
3.7.2.1	children	12
3.7.2.2	value	12
3.8	trie_t Struct Reference	12
3.8.1	Detailed Description	13
3.8.2	Member Data Documentation	13
3.8.2.1	member_size	13
3.8.2.2	root	13
3.8.2.3	size	13
3.9	vector_t Struct Reference	13
3.9.1	Member Data Documentation	13
3.9.1.1	buffer_size	13
3.9.1.2	data	14
3.9.1.3	member_size	14
3.9.1.4	size	14

4 File Documentation	15
4.1 include/gerror.h File Reference	15
4.1.1 Typedef Documentation	16
4.1.1.1 gerror_t	16
4.1.2 Enumeration Type Documentation	16
4.1.2.1 gerror_t	16
4.1.3 Function Documentation	16
4.1.3.1 gerror_to_str()	16
4.2 include/graph.h File Reference	17
4.2.1 Typedef Documentation	18
4.2.1.1 graph_t	18
4.2.2 Function Documentation	18
4.2.2.1 graph_add_edge()	18
4.2.2.2 graph_create()	18
4.2.2.3 graph_destroy()	19
4.2.2.4 graph_get_label_at()	19
4.2.2.5 graph_set_label_at()	19
4.3 include/priority_queue.h File Reference	20
4.3.1 Typedef Documentation	21
4.3.1.1 compare_function	21
4.3.1.2 pqueue_t	22
4.3.1.3 priority_queue_t	22
4.3.2 Enumeration Type Documentation	22
4.3.2.1 queue_priority_t	22
4.3.3 Function Documentation	22
4.3.3.1 pqueue_add()	22
4.3.3.2 pqueue_create()	22
4.3.3.3 pqueue_destroy()	24
4.3.3.4 pqueue_extract()	24
4.3.3.5 pqueue_max_priority()	25

4.3.3.6	<code>pqueue_set_compare_function()</code>	25
4.4	<code>include/queue.h</code> File Reference	25
4.4.1	Typedef Documentation	27
4.4.1.1	<code>qnode_t</code>	27
4.4.1.2	<code>queue_t</code>	27
4.4.2	Function Documentation	27
4.4.2.1	<code>queue_create()</code>	27
4.4.2.2	<code>queue_dequeue()</code>	27
4.4.2.3	<code>queue_destroy()</code>	28
4.4.2.4	<code>queue_enqueue()</code>	28
4.4.2.5	<code>queue_remove()</code>	28
4.5	<code>include/stack.h</code> File Reference	29
4.5.1	Typedef Documentation	30
4.5.1.1	<code>snode_t</code>	30
4.5.1.2	<code>stack_t</code>	30
4.5.2	Function Documentation	30
4.5.2.1	<code>stack_create()</code>	30
4.5.2.2	<code>stack_destroy()</code>	31
4.5.2.3	<code>stack_pop()</code>	31
4.5.2.4	<code>stack_push()</code>	31
4.6	<code>include/trie.h</code> File Reference	32
4.6.1	Macro Definition Documentation	33
4.6.1.1	<code>NBYTE</code>	33
4.6.2	Typedef Documentation	33
4.6.2.1	<code>tnode_t</code>	33
4.6.2.2	<code>trie_t</code>	33
4.6.3	Function Documentation	33
4.6.3.1	<code>trie_add_element()</code>	33
4.6.3.2	<code>trie_create()</code>	34
4.6.3.3	<code>trie_destroy()</code>	34

4.6.3.4	trie_get_element()	34
4.6.3.5	trie_get_node_or_allocate()	35
4.6.3.6	trie_remove_element()	35
4.6.3.7	trie_set_element()	35
4.7	include/vector.h File Reference	36
4.7.1	Typedef Documentation	37
4.7.1.1	vector_t	37
4.7.2	Function Documentation	37
4.7.2.1	vector_add()	37
4.7.2.2	vector_append()	37
4.7.2.3	vector_append_buffer()	38
4.7.2.4	vector_at()	38
4.7.2.5	vector_create()	39
4.7.2.6	vector_destroy()	39
4.7.2.7	vector_get_min_buf_siz()	39
4.7.2.8	vector_ptr_at()	40
4.7.2.9	vector_resize_buffer()	40
4.7.2.10	vector_set_elem_at()	40
4.7.2.11	vector_set_min_buf_siz()	41
4.8	src/gerror.c File Reference	41
4.8.1	Function Documentation	42
4.8.1.1	gerror_to_str()	42
4.8.2	Variable Documentation	42
4.8.2.1	gerror_to_string	42
4.9	src/graph.c File Reference	42
4.9.1	Function Documentation	43
4.9.1.1	graph_add_edge()	43
4.9.1.2	graph_create()	43
4.9.1.3	graph_destroy()	44
4.9.1.4	graph_get_label_at()	44

4.9.1.5	<code>graph_set_label_at()</code>	44
4.10	<code>src/priority_queue.c</code> File Reference	45
4.10.1	Macro Definition Documentation	46
4.10.1.1	<code>LEFT</code>	46
4.10.1.2	<code>PARENT</code>	46
4.10.1.3	<code>RIGHT</code>	46
4.10.2	Function Documentation	46
4.10.2.1	<code>default_compare_function()</code>	46
4.10.2.2	<code>max_heapify()</code>	46
4.10.2.3	<code>nswap()</code>	46
4.10.2.4	<code>pqueue_add()</code>	46
4.10.2.5	<code>pqueue_create()</code>	47
4.10.2.6	<code>pqueue_destroy()</code>	47
4.10.2.7	<code>pqueue_extract()</code>	48
4.10.2.8	<code>pqueue_max_priority()</code>	48
4.10.2.9	<code>pqueue_set_compare_function()</code>	48
4.11	<code>src/queue.c</code> File Reference	49
4.11.1	Function Documentation	50
4.11.1.1	<code>queue_create()</code>	50
4.11.1.2	<code>queue_dequeue()</code>	50
4.11.1.3	<code>queue_destroy()</code>	50
4.11.1.4	<code>queue_enqueue()</code>	51
4.11.1.5	<code>queue_remove()</code>	51
4.12	<code>src/stack.c</code> File Reference	51
4.12.1	Function Documentation	52
4.12.1.1	<code>stack_create()</code>	52
4.12.1.2	<code>stack_destroy()</code>	53
4.12.1.3	<code>stack_pop()</code>	53
4.12.1.4	<code>stack_push()</code>	53
4.13	<code>src/trie.c</code> File Reference	54

4.13.1	Function Documentation	54
4.13.1.1	node_at()	54
4.13.1.2	trie_add_element()	55
4.13.1.3	trie_create()	55
4.13.1.4	trie_destroy()	55
4.13.1.5	trie_destroy_tnode()	55
4.13.1.6	trie_get_element()	56
4.13.1.7	trie_get_node_or_allocate()	56
4.13.1.8	trie_remove_element()	56
4.13.1.9	trie_set_element()	57
4.14	src/vector.c File Reference	57
4.14.1	Macro Definition Documentation	58
4.14.1.1	VECTOR_MIN_SIZ	58
4.14.2	Function Documentation	58
4.14.2.1	vector_add()	58
4.14.2.2	vector_append()	58
4.14.2.3	vector_append_buffer()	60
4.14.2.4	vector_at()	60
4.14.2.5	vector_create()	61
4.14.2.6	vector_destroy()	61
4.14.2.7	vector_get_min_buf_siz()	61
4.14.2.8	vector_ptr_at()	62
4.14.2.9	vector_resize_buffer()	62
4.14.2.10	vector_set_elem_at()	62
4.14.2.11	vector_set_min_buf_siz()	63
4.14.3	Variable Documentation	63
4.14.3.1	vector_min_siz	63

Chapter 1

Class Index

1.1 Class List

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

graph_t	5
priority_queue_t	6
qnode_t	7
queue_t	8
snode_t	9
stack_t	10
tnode_t	11
trie_t	12
vector_t	13

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

include/gerror.h	15
include/graph.h	17
include/priority_queue.h	20
include/queue.h	25
include/stack.h	29
include/trie.h	32
include/vector.h	36
src/gerror.c	41
src/graph.c	42
src/priority_queue.c	45
src/queue.c	49
src/stack.c	51
src/trie.c	54
src/vector.c	57

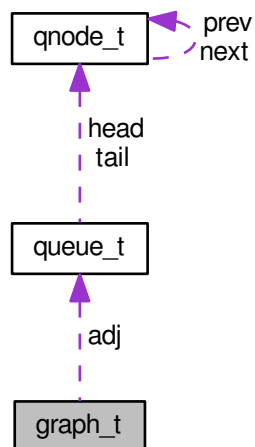
Chapter 3

Class Documentation

3.1 graph_t Struct Reference

```
#include <graph.h>
```

Collaboration diagram for graph_t:



Public Attributes

- `size_t` [V](#)
- `size_t` [E](#)
- `size_t` [member_size](#)
- `struct` [queue_t](#) * [adj](#)
- `void` * [label](#)

3.1.1 Detailed Description

Graph structure and elements.

3.1.2 Member Data Documentation

3.1.2.1 adj

```
struct queue\_t* graph_t::adj
```

3.1.2.2 E

```
size_t graph_t::E
```

3.1.2.3 label

```
void* graph_t::label
```

3.1.2.4 member_size

```
size_t graph_t::member_size
```

3.1.2.5 V

```
size_t graph_t::V
```

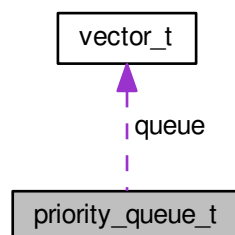
The documentation for this struct was generated from the following file:

- [include/graph.h](#)

3.2 [priority_queue_t](#) Struct Reference

```
#include <priority\_queue.h>
```

Collaboration diagram for [priority_queue_t](#):



Public Attributes

- `size_t` [size](#)
- `size_t` [member_size](#)
- `compare_function` [compare](#)
- `void *` [compare_argument](#)
- `struct` [vector_t](#) [queue](#)

3.2.1 Member Data Documentation

3.2.1.1 `compare`

`compare_function` `priority_queue_t::compare`

3.2.1.2 `compare_argument`

`void*` `priority_queue_t::compare_argument`

3.2.1.3 `member_size`

`size_t` `priority_queue_t::member_size`

3.2.1.4 `queue`

`struct` [vector_t](#) `priority_queue_t::queue`

3.2.1.5 `size`

`size_t` `priority_queue_t::size`

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

- `include/priority_queue.h`

3.3 qnode_t Struct Reference

```
#include <queue.h>
```

Collaboration diagram for `qnode_t`:



Public Attributes

- struct `qnode_t` * `next`
- struct `qnode_t` * `prev`
- void * `data`

3.3.1 Detailed Description

queue node.

3.3.2 Member Data Documentation

3.3.2.1 data

```
void* qnode_t::data
```

3.3.2.2 next

```
struct qnode_t* qnode_t::next
```

3.3.2.3 prev

```
struct qnode_t* qnode_t::prev
```

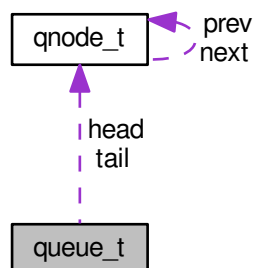
The documentation for this struct was generated from the following file:

- include/`queue.h`

3.4 queue_t Struct Reference

```
#include <queue.h>
```

Collaboration diagram for `queue_t`:



Public Attributes

- `size_t` [size](#)
- `size_t` [member_size](#)
- `struct qnode_t *` [head](#)
- `struct qnode_t *` [tail](#)

3.4.1 Detailed Description

Represents a queue structure.

3.4.2 Member Data Documentation

3.4.2.1 head

```
struct qnode_t* queue_t::head
```

3.4.2.2 member_size

```
size_t queue_t::member_size
```

3.4.2.3 size

```
size_t queue_t::size
```

3.4.2.4 tail

```
struct qnode_t* queue_t::tail
```

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

- `include/queue.h`

3.5 snode_t Struct Reference

```
#include <stack.h>
```

Collaboration diagram for `snode_t`:



Public Attributes

- struct [snode_t](#) * [next](#)
- struct [snode_t](#) * [prev](#)
- void * [data](#)

3.5.1 Detailed Description

node of a stack

3.5.2 Member Data Documentation

3.5.2.1 data

```
void* snode_t::data
```

3.5.2.2 next

```
struct snode\_t* snode_t::next
```

3.5.2.3 prev

```
struct snode\_t* snode_t::prev
```

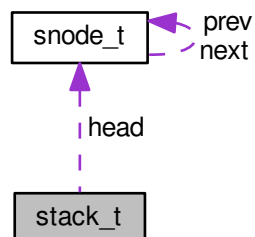
The documentation for this struct was generated from the following file:

- include/[stack.h](#)

3.6 stack_t Struct Reference

```
#include <stack.h>
```

Collaboration diagram for `stack_t`:



Public Attributes

- [size_t](#) [size](#)
- [size_t](#) [member_size](#)
- [struct](#) [snode_t](#) * [head](#)

3.6.1 Detailed Description

represents the stack structure.

3.6.2 Member Data Documentation

3.6.2.1 head

```
struct snode\_t* stack_t::head
```

3.6.2.2 member_size

```
size_t stack_t::member_size
```

3.6.2.3 size

```
size_t stack_t::size
```

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

- [include/stack.h](#)

3.7 tnode_t Struct Reference

```
#include <trie.h>
```

Collaboration diagram for tnode_t:



Public Attributes

- void * [value](#)
- struct [tnode_t](#) * [children](#) [NBYTE]

3.7.1 Detailed Description

node of a [trie_t](#) element.

3.7.2 Member Data Documentation

3.7.2.1 children

```
struct tnode\_t* tnode_t::children[NBYTE]
```

3.7.2.2 value

```
void* tnode_t::value
```

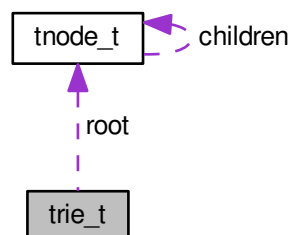
The documentation for this struct was generated from the following file:

- include/[trie.h](#)

3.8 [trie_t](#) Struct Reference

```
#include <trie.h>
```

Collaboration diagram for [trie_t](#):



Public Attributes

- `size_t` [size](#)
- `size_t` [member_size](#)
- `struct tnode_t` [root](#)

3.8.1 Detailed Description

Represents the trie structure.

3.8.2 Member Data Documentation

3.8.2.1 member_size

```
size_t trie_t::member_size
```

3.8.2.2 root

```
struct tnode_t trie_t::root
```

3.8.2.3 size

```
size_t trie_t::size
```

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

- `include/trie.h`

3.9 vector_t Struct Reference

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

Public Attributes

- `void *` [data](#)
- `size_t` [size](#)
- `size_t` [buffer_size](#)
- `size_t` [member_size](#)

3.9.1 Member Data Documentation

3.9.1.1 buffer_size

```
size_t vector_t::buffer_size
```

3.9.1.2 data

```
void* vector_t::data
```

3.9.1.3 member_size

```
size_t vector_t::member_size
```

3.9.1.4 size

```
size_t vector_t::size
```

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

- [include/vector.h](#)

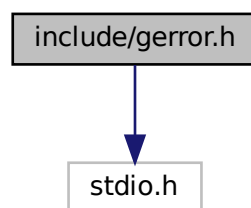
Chapter 4

File Documentation

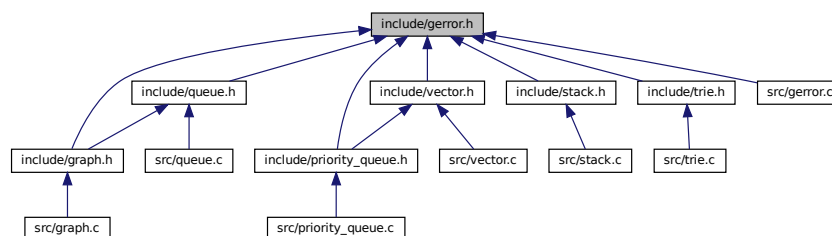
4.1 include/gerror.h File Reference

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

Include dependency graph for gerror.h:



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



Typedefs

- typedef enum [gerror_t](#) [gerror_t](#)

Enumerations

- enum `gerror_t` {
`GERROR_OK`, `GERROR_NULL_STRUCTURE`, `GERROR_NULL_HEAD`, `GERROR_NULL_NODE`,
`GERROR_NULL_POINTER_TO_BUFFER`, `GERROR_TRY_REMOVE_EMPTY_STRUCTURE`, `GERROR_TRY_ADD_EDGE_NO_VERTEX`, `GERROR_ACCESS_OUT_OF_BOUND`,
`GERROR_INCOMPATIBLE_VECTOR_APPEND_SIZE`, `GERROR_N_ERROR` }

Functions

- char * `gerror_to_str` (`gerror_t` g)

4.1.1 Typedef Documentation

4.1.1.1 `gerror_t`

```
typedef enum gerror_t gerror_t
```

4.1.2 Enumeration Type Documentation

4.1.2.1 `gerror_t`

```
enum gerror_t
```

Enumerator

<code>GERROR_OK</code>	
<code>GERROR_NULL_STRUCTURE</code>	
<code>GERROR_NULL_HEAD</code>	
<code>GERROR_NULL_NODE</code>	
<code>GERROR_NULL_POINTER_TO_BUFFER</code>	
<code>GERROR_TRY_REMOVE_EMPTY_STRUCTURE</code>	
<code>GERROR_TRY_ADD_EDGE_NO_VERTEX</code>	
<code>GERROR_ACCESS_OUT_OF_BOUND</code>	
<code>GERROR_INCOMPATIBLE_VECTOR_APPEND_SIZE</code>	
<code>GERROR_N_ERROR</code>	

4.1.3 Function Documentation

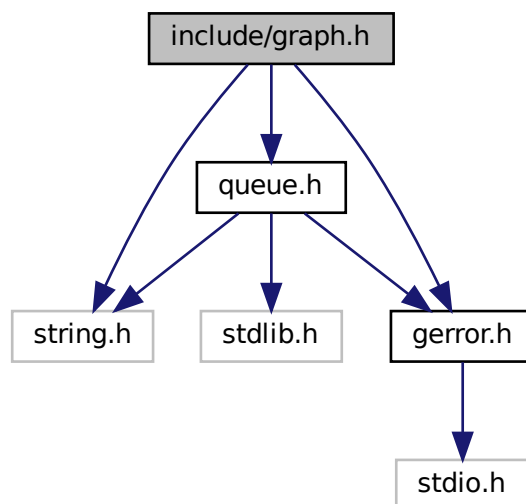
4.1.3.1 `gerror_to_str()`

```
char* gerror_to_str (  

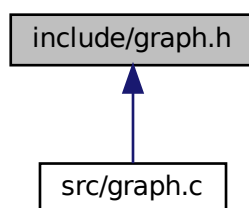
    gerror_t g )
```

4.2 include/graph.h File Reference

```
#include <string.h>
#include "gerror.h"
#include "queue.h"
Include dependency graph for graph.h:
```



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



Classes

- struct [graph_t](#)

Typedefs

- typedef struct [graph_t](#) [graph_t](#)

Functions

- [gerror_t graph_create](#) ([graph_t](#) *g, [size_t](#) size, [size_t](#) member_size)
- [gerror_t graph_add_edge](#) ([graph_t](#) *g, [size_t](#) from, [size_t](#) to)
- [gerror_t graph_get_label_at](#) ([graph_t](#) *g, [size_t](#) index, void *label)
- [gerror_t graph_set_label_at](#) ([graph_t](#) *g, [size_t](#) index, void *label)
- [gerror_t graph_destroy](#) ([graph_t](#) *g)

4.2.1 Typedef Documentation

4.2.1.1 graph_t

```
typedef struct graph_t graph_t
```

Graph structure and elements.

4.2.2 Function Documentation

4.2.2.1 graph_add_edge()

```
gerror_t graph_add_edge (
    graph_t * g,
    size_t from,
    size_t to )
```

Adds an edge on the graph *g* from the vertex *from* to the vertex *to*. Where *from* and *to* are indexes of these vertex.

Parameters

<i>g</i>	pointer to a graph structure;
<i>from</i>	index of the first vertex;
<i>to</i>	index of the incident vertex.

Returns

ERROR_OK in case of success operation; ERROR_TRY_ADD_EDGE_NO_VERTEX in case that *from* or *to* not exists in the graph

4.2.2.2 graph_create()

```
gerror_t graph_create (
    graph_t * g,
    size_t size,
    size_t member_size )
```

Creates a graph and populates the previous allocated structure pointed by *g*;

Parameters

<i>g</i>	pointer to a graph structure;
<i>member_size</i>	size of the elements that will be indexed by <i>g</i>

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case *g* is a NULL

4.2.2.3 graph_destroy()

```
gerror_t graph_destroy (  
    graph_t * g )
```

Deallocates the structures in *g*. This function WILL NOT deallocate the pointer *g*.

Parameters

<i>g</i>	pointer to a graph structure;
----------	-------------------------------

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case *g* is a NULL

4.2.2.4 graph_get_label_at()

```
gerror_t graph_get_label_at (  
    graph_t * g,  
    size_t index,  
    void * label )
```

Gets the label of the vertex in the *index* position of the graph *g*.

Parameters

<i>g</i>	pointer to a graph structure;
<i>index</i>	index of the vertex;
<i>label</i>	pointer to the memory allocated that will be write with the label in <i>index</i>

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case *g* is a NULL

4.2.2.5 graph_set_label_at()

```
gerror_t graph_set_label_at (  
    graph_t * g,
```

```
size_t index,  
void * label )
```

Sets the label at the `index` to `label`.

Parameters

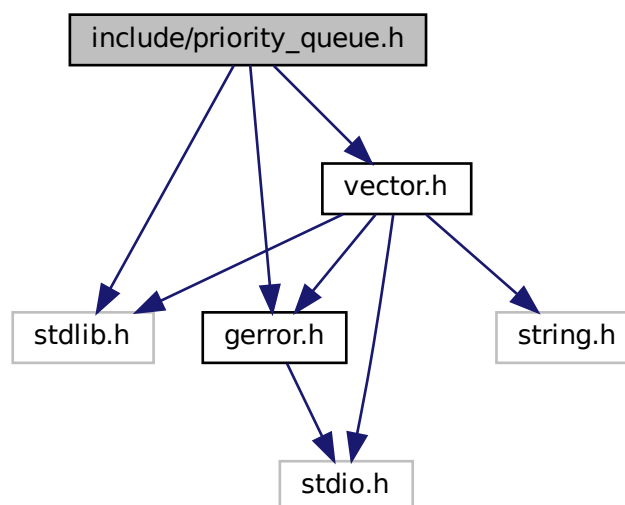
<i>g</i>	pointer to a graph structure;
<i>index</i>	index of the vertex;
<i>label</i>	the new label of the vertex positioned in <i>index</i>

Returns

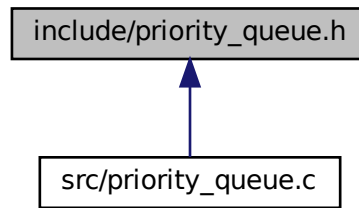
`ERROR_OK` in case of success operation; `ERROR_ACCESS_OUT_OF_BOUND` in case that `index` is out of bound

4.3 include/priority_queue.h File Reference

```
#include <stdlib.h>  
#include "gerror.h"  
#include "vector.h"  
Include dependency graph for priority_queue.h:
```



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



Classes

- struct [priority_queue_t](#)

Typedefs

- typedef int(* [compare_function](#)) (void *a, void *b, void *arg)
- typedef struct [priority_queue_t](#) [priority_queue_t](#)
- typedef struct [priority_queue_t](#) [pqueue_t](#)

Enumerations

- enum [queue_priority_t](#) { [G_PQUEUE_FIRST_PRIORITY](#) = -1, [G_PQUEUE_EQUAL_PRIORITY](#), [G_PQUEUE_SECOND_PRIORITY](#) }

Functions

- [gerror_t](#) [pqueue_create](#) ([pqueue_t](#) *p, [size_t](#) member_size)
- [gerror_t](#) [pqueue_destroy](#) ([pqueue_t](#) *p)
- [gerror_t](#) [pqueue_set_compare_function](#) ([pqueue_t](#) *p, [compare_function](#) function, void *argument)
- [gerror_t](#) [pqueue_add](#) ([pqueue_t](#) *p, void *e)
- [gerror_t](#) [pqueue_max_priority](#) ([pqueue_t](#) *p, void *e)
- [gerror_t](#) [pqueue_extract](#) ([pqueue_t](#) *p, void *e)

4.3.1 Typedef Documentation

4.3.1.1 [compare_function](#)

```
typedef int(* compare\_function) (void *a, void *b, void *arg)
```

4.3.1.2 pqueue_t

```
typedef struct priority_queue_t pqueue_t
```

4.3.1.3 priority_queue_t

```
typedef struct priority_queue_t priority_queue_t
```

4.3.2 Enumeration Type Documentation

4.3.2.1 queue_priority_t

```
enum queue_priority_t
```

Enumerator

G_PQUEUE_FIRST_PRIORITY	
G_PQUEUE_EQUAL_PRIORITY	
G_PQUEUE_SECOND_PRIORITY	

4.3.3 Function Documentation

4.3.3.1 pqueue_add()

```
gerror_t pqueue_add (
    pqueue_t * p,
    void * e )
```

Adds an element in the queue and max heap the queue. TODO: A more detailed description of pqueue_add.

Parameters

<i>p</i>	previous allocated pqueue_t struct
<i>e</i>	the element to be added

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case *t* is a NULL

4.3.3.2 pqueue_create()

```
gerror_t pqueue_create (
    pqueue_t * p,
    size_t member_size )
```


Populates the `p` structure and initialize it. A priority queue needs a `compare_function`. The default function will only work for `char`, `int` and `long`. If you need a `double` or `float` you need to implement the compare function and set with the function `pqueue_set_compare_function`

Parameters

<i>p</i>	previous allocated pqueue_t struct
<i>member_size</i>	size in bytes of the indexed elements
<i>function</i>	comparison function callback that has the following prototype: int compare(void* a, void* b) the a and b are the arguments returns -1 if a has priority BIG than B returns 0 if a has priority EQUAL than B return 1 if a has priority LE

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case p is a NULL

4.3.3.3 pqueue_destroy()

```
gerror_t pqueue_destroy (
    pqueue_t * p )
```

Destroy (i.e. desallocates) the p structure fields. TODO: A more daitailed description of pqueue_destroy.

Parameters

<i>p</i>	previous allocated pqueue_t struct
----------	------------------------------------

Returns

TODO

4.3.3.4 pqueue_extract()

```
gerror_t pqueue_extract (
    pqueue_t * p,
    void * e )
```

Extracts the highest priority element in the queue and writes in e pointer.

Parameters

<i>p</i>	previous allocated pqueue_t struct
<i>e</i>	pointer to previous allocated variable

Returns

GERROR_OK in case of success operation; GERROR_ACESS_OUT_OF_BOUND in case the queue is empty GERROR_NULL_STRUCURE in case t is a NULL

4.3.3.5 pqueue_max_priority()

```
gerror_t pqueue_max_priority (
    pqueue_t * p,
    void * e )
```

Returns and does not remove the highest priority of the queue. TODO: A more detailed description of pqueue_max_priority.

Parameters

<i>p</i>	previous allocated pqueue_t struct
<i>e</i>	pointer to previous allocated variable with member_size size that will receive a copy of the highest priority element of the queue.

Returns

GERROR_OK in case of success operation; GERROR_ACESS_OUT_OF_BOUND in case the queue is empty GERROR_NULL_STRUCURE in case *t* is a NULL

4.3.3.6 pqueue_set_compare_function()

```
gerror_t pqueue_set_compare_function (
    pqueue_t * p,
    compare_function function,
    void * argument )
```

Change the default comparison function of the priority queue *p* by *function* with the argument *argument*.

Parameters

<i>p</i>	previous allocated pqueue_t struct
<i>function</i>	comparison function callback that has the following prototype: int compare(void* a, void* b) the a and b are the arguments returns -1 if a has priority BIG than B returns 0 if a has priority EQUAL than B return 1 if a has priority LE
<i>argument</i>	allocated pqueue_t struct

Returns

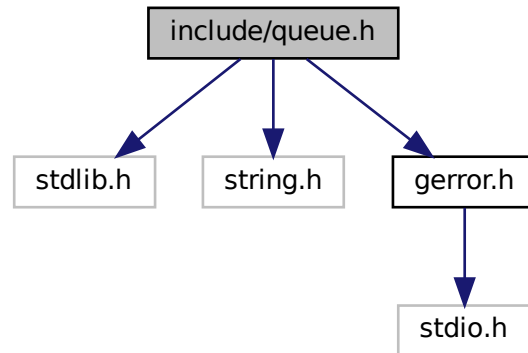
GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case *t* is a NULL

4.4 include/queue.h File Reference

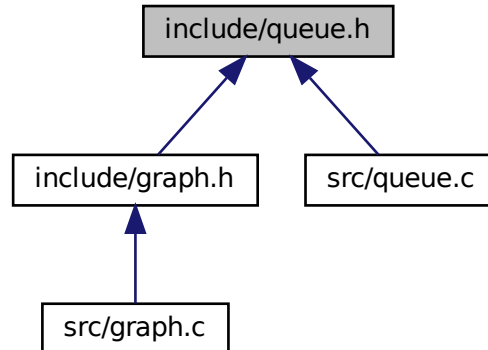
```
#include <stdlib.h>
#include <string.h>
```

```
#include "gerror.h"
```

Include dependency graph for queue.h:



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



Classes

- struct `qnode_t`
- struct `queue_t`

Typedefs

- typedef struct `qnode_t` `qnode_t`
- typedef struct `queue_t` `queue_t`

Functions

- [gerror_t queue_create](#) (struct [queue_t](#) *q, size_t member_size)
- [gerror_t queue_enqueue](#) (struct [queue_t](#) *q, void *e)
- [gerror_t queue_dequeue](#) (struct [queue_t](#) *q, void *e)
- [gerror_t queue_destroy](#) (struct [queue_t](#) *q)
- [gerror_t queue_remove](#) (struct [queue_t](#) *q, struct [qnode_t](#) *node, void *e)

4.4.1 Typedef Documentation

4.4.1.1 qnode_t

```
typedef struct qnode_t qnode_t
```

queue node.

4.4.1.2 queue_t

```
typedef struct queue_t queue_t
```

Represents a queue structure.

4.4.2 Function Documentation

4.4.2.1 queue_create()

```
gerror_t queue_create (
    struct queue_t * q,
    size_t member_size )
```

Creates a queue and populates the previous allocated structure pointed by *q*;

Parameters

<i>q</i>	pointer to a queue structure;
<i>member_size</i>	size of the elements that will be indexed by <i>q</i>

Returns

ERROR_OK in case of success operation; ERROR_NULL_STRUCURE in case *q* is a NULL pointer

4.4.2.2 queue_dequeue()

```
gerror_t queue_dequeue (
    struct queue_t * q,
    void * e )
```

Dequeues the first element of the queue *q*

Parameters

<i>q</i>	pointer to a queue structure;
<i>e</i>	pointer to the previous allocated element memory that will be write with de dequeued element.

Returns

GERROR_OK in case of success operation; GERROR_NULL_HEAD in case that the head `q->head` is a null pointer. GERROR_NULL_STRUCURE in case `q` is a NULL pointer GERROR_TRY_REMOVE_EMPTY↵Y_STRUCTURE in case that `q` has no element.

4.4.2.3 queue_destroy()

```
gerror_t queue_destroy (
    struct queue_t * q )
```

Deallocate the nodes of the queue `q`. This function WILL NOT deallocate the pointer `q`.

Parameters

<i>q</i>	pointer to a queue structure;
----------	-------------------------------

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `q` is a NULL pointer

4.4.2.4 queue_enqueue()

```
gerror_t queue_enqueue (
    struct queue_t * q,
    void * e )
```

Enqueues the element pointed by `e` in the queue `q`.

Parameters

<i>q</i>	pointer to a queue structure;
<i>e</i>	pointer to the element that will be indexed by <code>q</code> .

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `q` is a NULL pointer

4.4.2.5 queue_remove()

```
gerror_t queue_remove (
    struct queue_t * q,
```

```
struct qnode_t * node,  
void * e )
```

Removes the element `node` of the queue `q`.

Parameters

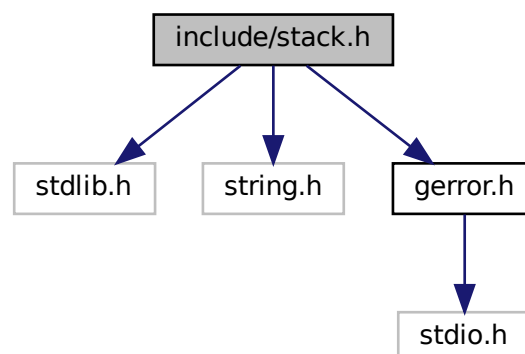
<i>q</i>	pointer to a queue structure;
<i>node</i>	element to be removed from the queue
<i>e</i>	pointer to the memory that will be write with the removed element

Returns

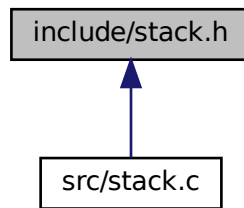
GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `q` is a NULL pointer GERROR_NULL_NODE in case `node` is NULL; GERROR_TRY_REMOVE_EMPTY_STRUCTURE in case that `q` has no element.

4.5 include/stack.h File Reference

```
#include <stdlib.h>  
#include <string.h>  
#include "gerror.h"  
Include dependency graph for stack.h:
```



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



Classes

- struct [snode_t](#)
- struct [stack_t](#)

Typedefs

- typedef struct [snode_t](#) [snode_t](#)
- typedef struct [stack_t](#) [stack_t](#)

Functions

- [gerror_t](#) [stack_create](#) (struct [stack_t](#) *q, size_t member_size)
- [gerror_t](#) [stack_push](#) (struct [stack_t](#) *q, void *e)
- [gerror_t](#) [stack_pop](#) (struct [stack_t](#) *q, void *e)
- [gerror_t](#) [stack_destroy](#) (struct [stack_t](#) *q)

4.5.1 Typedef Documentation

4.5.1.1 snode_t

```
typedef struct snode\_t snode\_t
```

node of a stack

4.5.1.2 stack_t

```
typedef struct stack\_t stack\_t
```

represents the stack structure.

4.5.2 Function Documentation

4.5.2.1 stack_create()

```
gerror\_t stack\_create (  
    struct stack\_t * s,  
    size_t member_size )
```

Creates a stack and populates the previous allocated structure pointed by *s*;

Parameters

<i>s</i>	pointer to a stack structure;
<i>member_size</i>	size of the elements that will be indexed by <i>s</i>

Returns

GERROR_OK in case of success operation; GERROR_NULL_ELEMENT in case that *e* is empty.

4.5.2.2 stack_destroy()

```
gerror_t stack_destroy (
    struct stack_t * s )
```

Deallocates the nodes of the structure pointed by *s*. This function WILL NOT deallocate the pointer *q*.

Parameters

<i>s</i>	pointer to a stack structure;
----------	-------------------------------

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCTURE in case *s* is a NULL

4.5.2.3 stack_pop()

```
gerror_t stack_pop (
    struct stack_t * s,
    void * e )
```

Pops the first element of the stack *s*.

Parameters

<i>s</i>	pointer to a stack structure;
<i>e</i>	pointer to the previous allocated element

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCTURE in case *s* is a NULL GERROR_NULL_HEAD in case that the head *s->head* GERROR_TRY_REMOVE_EMPTY_STRUCTURE in case that *s* is empty

4.5.2.4 stack_push()

```
gerror_t stack_push (
    struct stack_t * s,
    void * e )
```

Add the element e in the beginning of the stack s .

Parameters

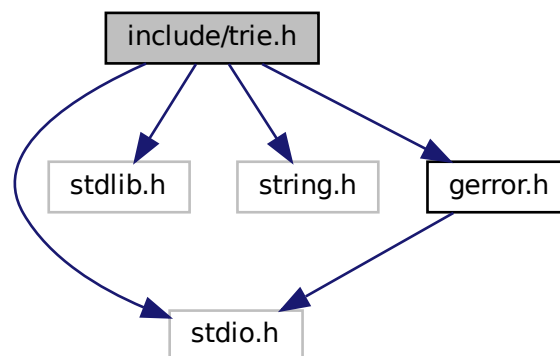
s	pointer to a stack structure;
e	pointer to the element that will be indexed by s .

Returns

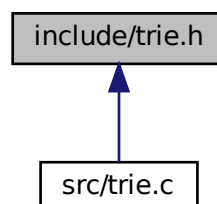
GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case s is a NULL

4.6 include/trie.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "gerror.h"
Include dependency graph for trie.h:
```



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



Classes

- struct [tnode_t](#)
- struct [trie_t](#)

Macros

- `#define` [NBYTE](#) (0x100)

Typedefs

- typedef struct [tnode_t](#) [tnode_t](#)
- typedef struct [trie_t](#) [trie_t](#)

Functions

- [gerror_t](#) [trie_create](#) (struct [trie_t](#) *t, size_t member_size)
- [gerror_t](#) [trie_destroy](#) (struct [trie_t](#) *t)
- [gerror_t](#) [trie_add_element](#) (struct [trie_t](#) *t, void *string, size_t size, void *elem)
- [gerror_t](#) [trie_remove_element](#) (struct [trie_t](#) *t, void *string, size_t size)
- [gerror_t](#) [trie_get_element](#) (struct [trie_t](#) *t, void *string, size_t size, void *elem)
- [gerror_t](#) [trie_set_element](#) (struct [trie_t](#) *t, void *string, size_t size, void *elem)
- [tnode_t](#) * [trie_get_node_or_allocate](#) (struct [trie_t](#) *t, void *string, size_t size)

4.6.1 Macro Definition Documentation

4.6.1.1 NBYTE

```
#define NBYTE (0x100)
```

4.6.2 Typedef Documentation

4.6.2.1 tnode_t

```
typedef struct tnode\_t tnode\_t
```

node of a [trie_t](#) element.

4.6.2.2 trie_t

```
typedef struct trie\_t trie\_t
```

Represents the trie structure.

4.6.3 Function Documentation

4.6.3.1 trie_add_element()

```
gerror\_t trie\_add\_element (  
    struct trie\_t * t,  
    void * string,  
    size_t size,  
    void * elem )
```

Adds the `elem` and maps it with the `string` with `size` `size`. This function overwrite any data left in the trie mapped with `string`.

Parameters

<i>t</i>	pointer to the trie structure;
<i>string</i>	pointer to the string of bytes to map elem;
<i>size</i>	size of the string of bytes
<i>elem</i>	pointer to the element to add

4.6.3.2 `trie_create()`

```
gerror_t trie_create (
    struct trie_t * t,
    size_t member_size )
```

Initialize structure `t` with `member_size` size. The `t` has to be allocated.

Parameters

<i>t</i>	pointer to the allocated struct <code>trie_t</code> ;
<i>member_size</i>	size in bytes of the indexed elements by the trie.

4.6.3.3 `trie_destroy()`

```
gerror_t trie_destroy (
    struct trie_t * t )
```

Destroy the members pointed by `t`. The structure is not freed.

Returns

`GERROR_OK` in case of success operation; `GERROR_NULL_STRUCURE` in case `t` is a `NULL`

4.6.3.4 `trie_get_element()`

```
gerror_t trie_get_element (
    struct trie_t * t,
    void * string,
    size_t size,
    void * elem )
```

Returns the element mapped by `string`. If the map does not exist, returns `NULL`.

Parameters

<i>t</i>	pointer to the structure;
<i>string</i>	pointer to the string of bytes to map elem;
<i>size</i>	size of the string of bytes.
<i>elem</i>	pointer to the memory allocated that will be write with the elem mapped by <code>string</code>

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case *t* is a NULL

4.6.3.5 trie_get_node_or_allocate()

```
tnode_t* trie_get_node_or_allocate (
    struct trie_t * t,
    void * string,
    size_t size )
```

4.6.3.6 trie_remove_element()

```
gerror_t trie_remove_element (
    struct trie_t * t,
    void * string,
    size_t size )
```

Removes the element mapped by *string*.

Parameters

<i>t</i>	pointer to the structure <code>trie_t</code> ;
<i>string</i>	pointer to the string of bytes to map elem;
<i>size</i>	size of the string of bytes.

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case *t* is a NULL GERROR↵_OUT_OF_BOUND the elem does not exist in *string* map

4.6.3.7 trie_set_element()

```
gerror_t trie_set_element (
    struct trie_t * t,
    void * string,
    size_t size,
    void * elem )
```

Sets the value mapped by *string*. Encapsulates the remove and add functions.

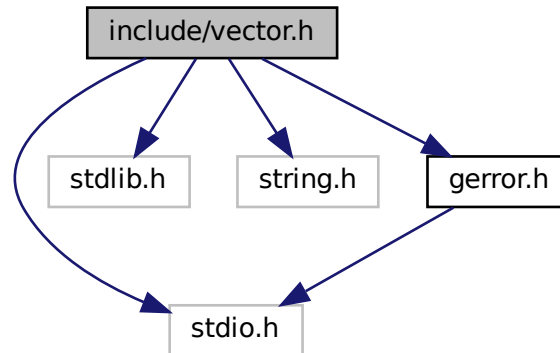
Parameters

<i>t</i>	pointer to the structure;
<i>string</i>	pointer to the string of bytes to map elem;
<i>size</i>	size of the string of bytes.
<i>elem</i>	pointer to the element to add

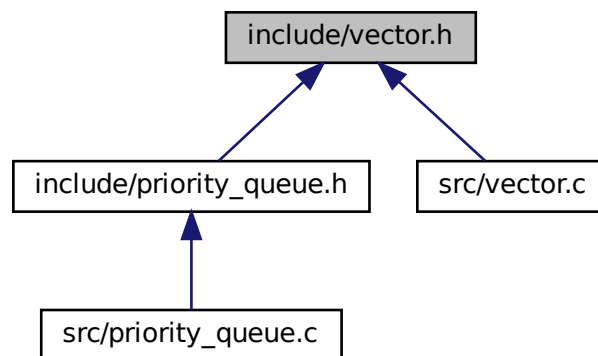
4.7 include/vector.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "gerror.h"
```

Include dependency graph for vector.h:



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



Classes

- struct [vector_t](#)

Typedefs

- typedef struct [vector_t](#) [vector_t](#)

Functions

- [gerror_t vector_create](#) ([vector_t](#) *v, [size_t](#) initial_size, [size_t](#) member_size)
- [gerror_t vector_destroy](#) ([vector_t](#) *v)
- [gerror_t vector_resize_buffer](#) ([vector_t](#) *v, [size_t](#) new_size)
- [gerror_t vector_at](#) ([vector_t](#) *v, [size_t](#) index, void *elem)
- [gerror_t vector_append](#) ([vector_t](#) *v0, [vector_t](#) *v1)
- [gerror_t vector_append_buffer](#) ([vector_t](#) *v, void *buffer, [size_t](#) size)
- void * [vector_ptr_at](#) ([vector_t](#) *v, [size_t](#) index)
- [gerror_t vector_set_elem_at](#) ([vector_t](#) *v, [size_t](#) index, void *elem)
- [gerror_t vector_add](#) ([vector_t](#) *v, void *elem)
- void [vector_set_min_buf_siz](#) ([size_t](#) new_min_buf_size)
- [size_t](#) [vector_get_min_buf_siz](#) (void)

4.7.1 Typedef Documentation

4.7.1.1 vector_t

```
typedef struct vector\_t vector\_t
```

4.7.2 Function Documentation

4.7.2.1 vector_add()

```
gerror\_t vector_add (
    vector\_t * v,
    void * elem )
```

adds the `elem` in the structure [vector_t](#) pointed by `v`.

Parameters

<code>v</code>	a pointer to vector_t
<code>elem</code>	the element to be add in <code>v</code>

Returns

`ERROR_OK` in case of success operation; `ERROR_NULL_STRUCTURE` in case `v` is a NULL pointer

4.7.2.2 vector_append()

```
gerror\_t vector_append (
    vector\_t * v0,
    vector\_t * v1 )
```

Appends the vector `v1` to the vector `v0`.

Parameters

<i>v0</i>	vector structure
<i>v1</i>	vector structure to be appended

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCTURE in case *v* is a NULL

4.7.2.3 vector_append_buffer()

```
gerror_t vector_append_buffer (
    vector_t * v,
    void * buffer,
    size_t size )
```

Appends *size* members pointed by *buffer* in *v*.

Parameters

<i>v</i>	a pointer to <code>vector_t</code> structure
<i>buffer</i>	pointer to the elements with the same size to <i>v->member_size</i> to be appended
<i>size</i>	number of elements pointed by the <i>buffer</i>

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCTURE in case *v* is a NULL

4.7.2.4 vector_at()

```
gerror_t vector_at (
    vector_t * v,
    size_t index,
    void * elem )
```

Get the element in the *index* position indexed by the `vector_t` structure pointed by *v*.

Parameters

<i>v</i>	a pointer to <code>vector_t</code>
<i>index</i>	index of the position
<i>elem</i>	pointer to a previous allocated memory that will receive the element

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case *v* is a NULL pointer

4.7.2.5 vector_create()

```
gerror_t vector_create (
    vector_t * v,
    size_t initial_buf_siz,
    size_t member_size )
```

Populate the `vector_t` structure pointed by `v` and allocates `member_size*initial_size` for initial buffer↵
_size.

Parameters

<code>v</code>	a pointer to <code>vector_t</code> structure already allocated;
<code>initial_buf_size</code>	number of the members of the initial allocated buffer;
<code>member_size</code>	size of every member indexed by <code>v</code> .

Returns

`GERROR_OK` in case of success operation; `GERROR_NULL_STRUCURE` in case `v` is a NULL pointer

4.7.2.6 vector_destroy()

```
gerror_t vector_destroy (
    vector_t * v )
```

Destroy the structure `vector_t` pointed by `v`.

Parameters

<code>v</code>	a pointer to <code>vector_t</code> structure
----------------	--

Returns

`GERROR_OK` in case of success operation; `GERROR_NULL_STRUCURE` in case `v` is a NULL pointer

4.7.2.7 vector_get_min_buf_siz()

```
size_t vector_get_min_buf_siz (
    void )
```

Returns the `vector_min_siz`: a private variable that holds the minimal number of elements that `vector_t` will index. This variable is important for avoid multiple small resizes in the `vector_t` container.

Returns

`vector_min_siz`

4.7.2.8 vector_ptr_at()

```
void* vector_ptr_at (
    vector_t * v,
    size_t index )
```

Calculate the pointer at `index` position.

Parameters

<i>v</i>	a pointer to <code>vector_t</code>
<i>index</i>	index of the pointer

Returns

a pointer to the `index` element NULL in case of out of bound

4.7.2.9 vector_resize_buffer()

```
gerror_t vector_resize_buffer (
    vector_t * v,
    size_t n_elements )
```

Resize the buffer in the `vector_t` structure pointed by `v`.

Parameters

<i>v</i>	a pointer to <code>vector_t</code> structure.
<i>new_size</i>	the new size of the <code>v</code>

4.7.2.10 vector_set_elem_at()

```
gerror_t vector_set_elem_at (
    vector_t * v,
    size_t index,
    void * elem )
```

set the element at `index` pointed by `v` with the element pointed by `elem`.

Parameters

<i>v</i>	a pointer to <code>vector_t</code>
<i>index</i>	index of the position
<i>elem</i>	the element to be set in <code>v</code>

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `v` is a NULL pointer

4.7.2.11 vector_set_min_buf_siz()

```
void vector_set_min_buf_siz (
    size_t new_min_buf_siz )
```

Set the `vector_min_siz`: a private variable that holds the minimal number of elements that `vector_t` will index. This variable is important for avoid multiple small resizes in the `vector_t` container.

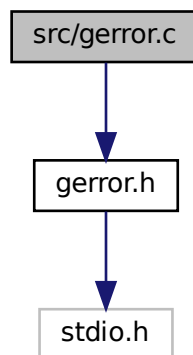
Parameters

<code>new_min_buf_siz</code>	the new size of <code>vector_min_siz</code>
------------------------------	---

4.8 src/gerror.c File Reference

```
#include "gerror.h"
```

Include dependency graph for gerror.c:

**Functions**

- `char * gerror_to_str (gerror_t g)`

Variables

- `char * gerror_to_string [GERROR_N_ERROR]`

4.8.1 Function Documentation

4.8.1.1 gerror_to_str()

```
char* gerror_to_str (
    gerror_t g )
```

4.8.2 Variable Documentation

4.8.2.1 gerror_to_string

```
char* gerror_to_string[GERROR_N_ERROR]
```

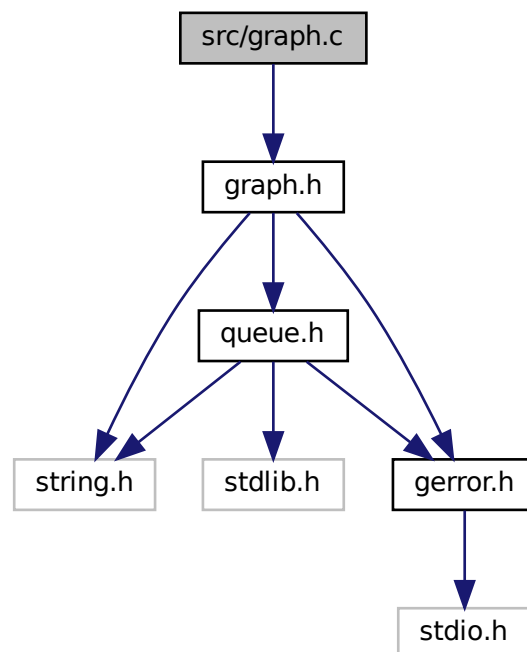
Initial value:

```
= {
    "Success",
    "Null pointer to structure",
    "Null pointer to the head of structure",
    "Null pointer to the node",
    "Null pointer to the buffer",
    "Attempt to remove an element but the structure is empty",
    "Attempt to add a edge with inexistent vertex",
    "Attempt to access a position out of the container or buffer",
    "Attempt to append two vectors with different member_size"
}
```

4.9 src/graph.c File Reference

```
#include "graph.h"
```

Include dependency graph for graph.c:



Functions

- [gerror_t graph_create](#) ([graph_t](#) *g, [size_t](#) size, [size_t](#) member_size)
- [gerror_t graph_add_edge](#) ([graph_t](#) *g, [size_t](#) from, [size_t](#) to)
- [gerror_t graph_get_label_at](#) ([graph_t](#) *g, [size_t](#) index, void *label)
- [gerror_t graph_set_label_at](#) ([graph_t](#) *g, [size_t](#) index, void *label)
- [gerror_t graph_destroy](#) ([graph_t](#) *g)

4.9.1 Function Documentation

4.9.1.1 graph_add_edge()

```
gerror_t graph_add_edge (
    graph_t * g,
    size_t from,
    size_t to )
```

Adds an edge on the graph *g* from the vertex *from* to the vertex *to*. Where *from* and *to* are indexes of these vertex.

Parameters

<i>g</i>	pointer to a graph structure;
<i>from</i>	index of the first vertex;
<i>to</i>	index of the incident vertex.

Returns

GERROR_OK in case of success operation; GERROR_TRY_ADD_EDGE_NO_VERTEX in case that *from* or *to* not exists in the graph

4.9.1.2 graph_create()

```
gerror_t graph_create (
    graph_t * g,
    size_t size,
    size_t member_size )
```

Creates a graph and populates the previous allocated structure pointed by *g*;

Parameters

<i>g</i>	pointer to a graph structure;
<i>member_size</i>	size of the elements that will be indexed by <i>g</i>

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case *g* is a NULL

4.9.1.3 graph_destroy()

```
gerror_t graph_destroy (
    graph_t * g )
```

Deallocates the structures in *g*. This function WILL NOT deallocate the pointer *g*.

Parameters

<i>g</i>	pointer to a graph structure;
----------	-------------------------------

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case *g* is a NULL

4.9.1.4 graph_get_label_at()

```
gerror_t graph_get_label_at (
    graph_t * g,
    size_t index,
    void * label )
```

Gets the label of the vertex in the *index* position of the graph *g*.

Parameters

<i>g</i>	pointer to a graph structure;
<i>index</i>	index of the vertex;
<i>label</i>	pointer to the memory allocated that will be write with the label in <i>index</i>

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case *g* is a NULL

4.9.1.5 graph_set_label_at()

```
gerror_t graph_set_label_at (
    graph_t * g,
    size_t index,
    void * label )
```

Sets the label at the *index* to *label*.

Parameters

<i>g</i>	pointer to a graph structure;
<i>index</i>	index of the vertex;
<i>label</i>	the new label of the vertex positioned in <i>index</i>

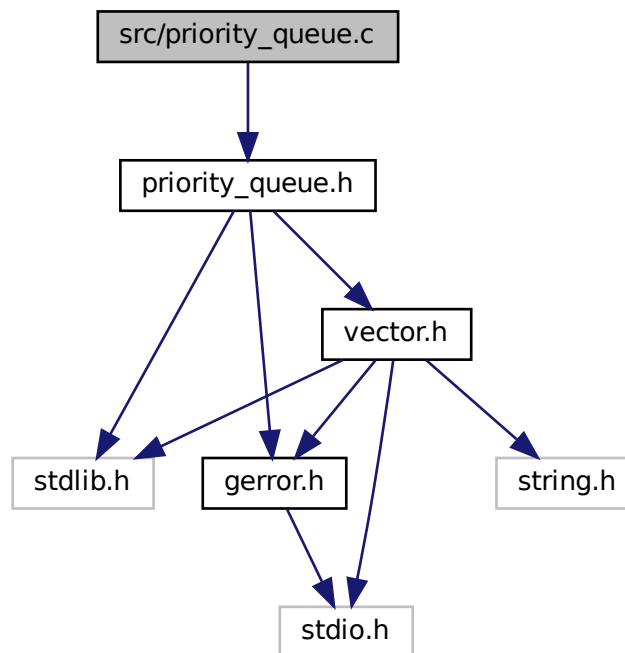
Returns

GERROR_OK in case of success operation; GERROR_ACCESS_OUT_OF_BOUND in case that `index` is out of bound

4.10 src/priority_queue.c File Reference

```
#include "priority_queue.h"
```

Include dependency graph for `priority_queue.c`:



Macros

- `#define PARENT(i) ((i-1)/2)`
- `#define LEFT(i) (((i+1)*2)-1)`
- `#define RIGHT(i) (LEFT(i)+1)`

Functions

- `void nswap (void *a, void *b, size_t n)`
- `int default_compare_function (void *a, void *b, void *arg)`
- `void max_heapify (pqueue_t *p, size_t i)`
- `gerror_t pqueue_create (pqueue_t *p, size_t member_size)`
- `gerror_t pqueue_destroy (pqueue_t *p)`
- `gerror_t pqueue_set_compare_function (pqueue_t *p, compare_function function, void *argument)`
- `gerror_t pqueue_add (pqueue_t *p, void *e)`
- `gerror_t pqueue_max_priority (pqueue_t *p, void *e)`
- `gerror_t pqueue_extract (pqueue_t *p, void *e)`

4.10.1 Macro Definition Documentation

4.10.1.1 LEFT

```
#define LEFT(  
    i )  ((i+1)*2)-1)
```

4.10.1.2 PARENT

```
#define PARENT(  
    i )  ((i-1)/2)
```

4.10.1.3 RIGHT

```
#define RIGHT(  
    i )  (LEFT(i)+1)
```

4.10.2 Function Documentation

4.10.2.1 default_compare_function()

```
int default_compare_function (  
    void * a,  
    void * b,  
    void * arg )
```

4.10.2.2 max_heapify()

```
void max_heapify (  
    pqueue_t * p,  
    size_t i )
```

4.10.2.3 nswap()

```
void nswap (  
    void * a,  
    void * b,  
    size_t n )
```

4.10.2.4 pqueue_add()

```
gerror_t pqueue_add (  
    pqueue_t * p,  
    void * e )
```

Adds an element in the queue and max heap the queue. TODO: A more detailed description of pqueue_add.

Parameters

<i>p</i>	previous allocated pqueue_t struct
<i>e</i>	the element to be added

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case *t* is a NULL

4.10.2.5 pqueue_create()

```
gerror_t pqueue_create (
    pqueue_t * p,
    size_t member_size )
```

Populates the *p* structure and initialize it. A priority queue needs a compare_function. The default function will only work for char, int and long. If you need a double or float you need to implement the compare function and set with the function pqueue_set_compare_function

Parameters

<i>p</i>	previous allocated pqueue_t struct
<i>member_size</i>	size in bytes of the indexed elements
<i>function</i>	comparison function callback that has the following prototype: int compare(void* a, void* b) the a and b are the arguments returns -1 if a has priority BIG than B returns 0 if a has priority EQUAL than B return 1 if a has priority LE

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case *p* is a NULL

4.10.2.6 pqueue_destroy()

```
gerror_t pqueue_destroy (
    pqueue_t * p )
```

Destroy (i.e. desallocates) the *p* structure fields. TODO: A more datailed description of pqueue_destroy.

Parameters

<i>p</i>	previous allocated pqueue_t struct
----------	------------------------------------

Returns

TODO

4.10.2.7 pqueue_extract()

```
gerror_t pqueue_extract (
    pqueue_t * p,
    void * e )
```

Extracts the highest priority element in the queue and writes in `e` pointer.

Parameters

<i>p</i>	previous allocated pqueue_t struct
<i>e</i>	pointer to previous allocated variable

Returns

GERROR_OK in case of success operation; GERROR_ACESS_OUT_OF_BOUND in case the queue is empty GERROR_NULL_STRUCURE in case `t` is a NULL

4.10.2.8 pqueue_max_priority()

```
gerror_t pqueue_max_priority (
    pqueue_t * p,
    void * e )
```

Returns and does not remove the highest priority of the queue. TODO: A more detailed description of pqueue_↔ max_priority.

Parameters

<i>p</i>	previous allocated pqueue_t struct
<i>e</i>	pointer to previous allocated variable with <code>member_size</code> size that will receive a copy of the highest priority element of the queue.

Returns

GERROR_OK in case of success operation; GERROR_ACESS_OUT_OF_BOUND in case the queue is empty GERROR_NULL_STRUCURE in case `t` is a NULL

4.10.2.9 pqueue_set_compare_function()

```
gerror_t pqueue_set_compare_function (
    pqueue_t * p,
    compare_function function,
    void * argument )
```

Change the default comparison function of the priority queue `p` by `function` with the argument `argument`.

Parameters

<i>p</i>	previous allocated pqueue_t struct
<i>function</i>	comparison function callback that has the following prototype: int compare(void* a, void* b) the a and b are the arguments returns -1 if a has priority BIG than B returns 0 if a has priority EQUAL than B return 1 if a has priority LE
<i>argument</i>	allocated pqueue_t struct

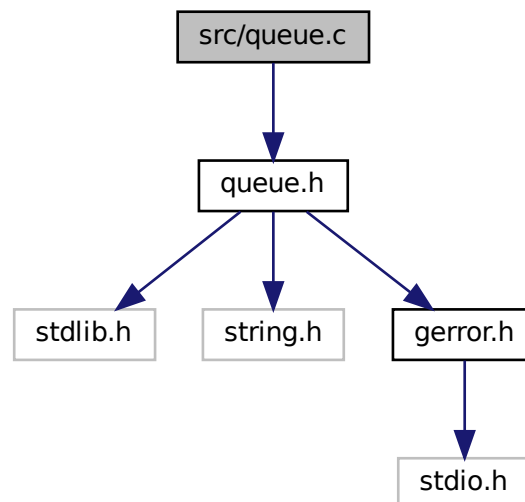
Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case t is a NULL

4.11 src/queue.c File Reference

```
#include "queue.h"
```

Include dependency graph for queue.c:



Functions

- [gerror_t queue_create](#) (struct [queue_t](#) *q, size_t member_size)
- [gerror_t queue_enqueue](#) (struct [queue_t](#) *q, void *e)
- [gerror_t queue_dequeue](#) (struct [queue_t](#) *q, void *e)
- [gerror_t queue_remove](#) (struct [queue_t](#) *q, struct [qnode_t](#) *node, void *e)
- [gerror_t queue_destroy](#) (struct [queue_t](#) *q)

4.11.1 Function Documentation

4.11.1.1 queue_create()

```
gerror_t queue_create (
    struct queue_t * q,
    size_t member_size )
```

Creates a queue and populates the previous allocated structure pointed by `q`;

Parameters

<code>q</code>	pointer to a queue structure;
<code>member_size</code>	size of the elements that will be indexed by <code>q</code>

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `q` is a NULL pointer

4.11.1.2 queue_dequeue()

```
gerror_t queue_dequeue (
    struct queue_t * q,
    void * e )
```

Dequeues the first element of the queue `q`

Parameters

<code>q</code>	pointer to a queue structure;
<code>e</code>	pointer to the previous allocated element memory that will be write with de dequeued element.

Returns

GERROR_OK in case of success operation; GERROR_NULL_HEAD in case that the head `q->head` is a null pointer. GERROR_NULL_STRUCURE in case `q` is a NULL pointer GERROR_TRY_REMOVE_EMPTY_STRUCTURE in case that `q` has no element.

4.11.1.3 queue_destroy()

```
gerror_t queue_destroy (
    struct queue_t * q )
```

Deallocate the nodes of the queue `q`. This function WILL NOT deallocate the pointer `q`.

Parameters

<code>q</code>	pointer to a queue structure;
----------------	-------------------------------

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case q is a NULL pointer

4.11.1.4 queue_enqueue()

```
gerror_t queue_enqueue (
    struct queue_t * q,
    void * e )
```

Enqueues the element pointed by e in the queue q .

Parameters

q	pointer to a queue structure;
e	pointer to the element that will be indexed by q .

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case q is a NULL pointer

4.11.1.5 queue_remove()

```
gerror_t queue_remove (
    struct queue_t * q,
    struct qnode_t * node,
    void * e )
```

Removes the element $node$ of the queue q .

Parameters

q	pointer to a queue structure;
$node$	element to be removed from the queue
e	pointer to the memory that will be write with the removed element

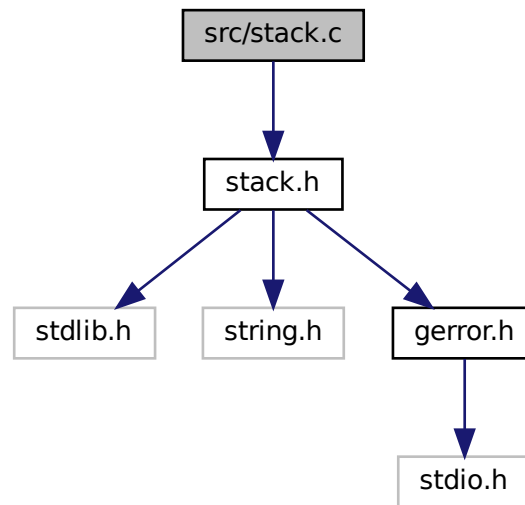
Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case q is a NULL pointer GERROR_NULL_NODE in case $node$ is NULL; GERROR_TRY_REMOVE_EMPTY_STRUCTURE in case that q has no element.

4.12 src/stack.c File Reference

```
#include "stack.h"
```

Include dependency graph for stack.c:



Functions

- [gerror_t stack_create](#) (struct [stack_t](#) *s, size_t member_size)
- [gerror_t stack_push](#) (struct [stack_t](#) *s, void *e)
- [gerror_t stack_pop](#) (struct [stack_t](#) *s, void *e)
- [gerror_t stack_destroy](#) (struct [stack_t](#) *s)

4.12.1 Function Documentation

4.12.1.1 stack_create()

```
gerror_t stack_create (
    struct stack\_t * s,
    size_t member_size )
```

Creates a stack and populates the previous allocated structure pointed by `s`;

Parameters

<code>s</code>	pointer to a stack structure;
<code>member_size</code>	size of the elements that will be indexed by <code>s</code>

Returns

`GERROR_OK` in case of success operation; `GERROR_NULL_ELEMENT` in case that `e` is empty.

4.12.1.2 `stack_destroy()`

```
gerror_t stack_destroy (
    struct stack_t * s )
```

Deallocates the nodes of the structure pointed by `s`. This function WILL NOT deallocate the pointer `q`.

Parameters

<code>s</code>	pointer to a stack structure;
----------------	-------------------------------

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `s` is a NULL

4.12.1.3 `stack_pop()`

```
gerror_t stack_pop (
    struct stack_t * s,
    void * e )
```

Pops the first element of the stack `s`.

Parameters

<code>s</code>	pointer to a stack structure;
<code>e</code>	pointer to the previous allocated element

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `s` is a NULL GERROR_NULL_HEAD in case that the head `s->head` GERROR_TRY_REMOVE_EMPTY_STRUCTURE in case that `s` is empty

4.12.1.4 `stack_push()`

```
gerror_t stack_push (
    struct stack_t * s,
    void * e )
```

Add the element `e` in the beginning of the stack `s`.

Parameters

<code>s</code>	pointer to a stack structure;
<code>e</code>	pointer to the element that will be indexed by <code>s</code> .

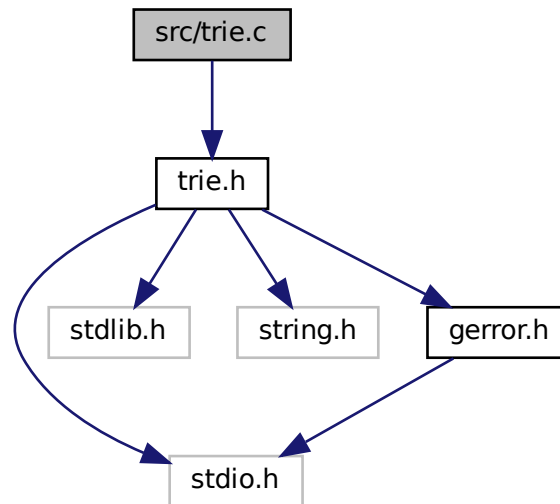
Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case *s* is a NULL

4.13 src/trie.c File Reference

```
#include "trie.h"
```

Include dependency graph for trie.c:



Functions

- `tnode_t * trie_get_node_or_allocate` (struct `trie_t` **t*, void **string*, size_t *size*)
- `tnode_t * node_at` (struct `trie_t` **t*, void **string*, size_t *size*)
- `gerror_t trie_create` (struct `trie_t` **t*, size_t *member_size*)
- void `trie_destroy_tnode` (struct `tnode_t` **node*)
- `gerror_t trie_destroy` (struct `trie_t` **t*)
- `gerror_t trie_add_element` (struct `trie_t` **t*, void **string*, size_t *size*, void **elem*)
- `gerror_t trie_remove_element` (struct `trie_t` **t*, void **string*, size_t *size*)
- `gerror_t trie_get_element` (struct `trie_t` **t*, void **string*, size_t *size*, void **elem*)
- `gerror_t trie_set_element` (struct `trie_t` **t*, void **string*, size_t *size*, void **elem*)

4.13.1 Function Documentation

4.13.1.1 node_at()

```
tnode_t* node_at (
    struct trie_t * t,
    void * string,
    size_t size )
```


4.13.1.2 `trie_add_element()`

```
gerror_t trie_add_element (
    struct trie\_t * t,
    void * string,
    size_t size,
    void * elem )
```

Adds the `elem` and maps it with the `string` with size `size`. This function overwrite any data left in the trie mapped with `string`.

Parameters

<i>t</i>	pointer to the trie structure;
<i>string</i>	pointer to the string of bytes to map <code>elem</code> ;
<i>size</i>	size of the string of bytes
<i>elem</i>	pointer to the element to add

4.13.1.3 `trie_create()`

```
gerror_t trie_create (
    struct trie\_t * t,
    size_t member_size )
```

Initialize structure `t` with `member_size` size. The `t` has to be allocated.

Parameters

<i>t</i>	pointer to the allocated struct trie_t ;
<i>member_size</i>	size in bytes of the indexed elements by the trie.

4.13.1.4 `trie_destroy()`

```
gerror_t trie_destroy (
    struct trie\_t * t )
```

Destroy the members pointed by `t`. The structure is not freed.

Returns

`ERROR_OK` in case of success operation; `ERROR_NULL_STRUCURE` in case `t` is a `NULL`

4.13.1.5 `trie_destroy_tnode()`

```
void trie_destroy_tnode (
    struct tnode\_t * node )
```

4.13.1.6 `trie_get_element()`

```
gerror_t trie_get_element (
    struct trie_t * t,
    void * string,
    size_t size,
    void * elem )
```

Returns the element mapped by `string`. If the map does not exist, returns NULL.

Parameters

<i>t</i>	pointer to the structure;
<i>string</i>	pointer to the string of bytes to map elem;
<i>size</i>	size of the string of bytes.
<i>elem</i>	pointer to the memory allocated that will be write with the elem mapped by <i>string</i>

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case *t* is a NULL

4.13.1.7 `trie_get_node_or_allocate()`

```
tnode_t* trie_get_node_or_allocate (
    struct trie_t * t,
    void * string,
    size_t size )
```

4.13.1.8 `trie_remove_element()`

```
gerror_t trie_remove_element (
    struct trie_t * t,
    void * string,
    size_t size )
```

Removes the element mapped by `string`.

Parameters

<i>t</i>	pointer to the structure <code>trie_t</code> ;
<i>string</i>	pointer to the string of bytes to map elem;
<i>size</i>	size of the string of bytes.

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case *t* is a NULL GERROR←
_OUT_OF_BOUND the *elem* does not exist in *string* map

4.13.1.9 trie_set_element()

```
gerror_t trie_set_element (
    struct trie_t * t,
    void * string,
    size_t size,
    void * elem )
```

Sets the value mapped by *string*. Encapsulates the remove and add functions.

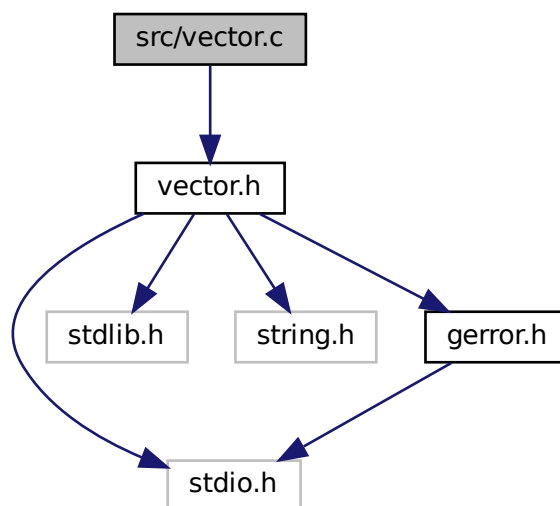
Parameters

<i>t</i>	pointer to the structure;
<i>string</i>	pointer to the string of bytes to map elem;
<i>size</i>	size of the string of bytes.
<i>elem</i>	pointer to the element to add

4.14 src/vector.c File Reference

```
#include "vector.h"
```

Include dependency graph for vector.c:



Macros

- `#define VECTOR_MIN_SIZ 8`

Functions

- `gerror_t vector_create (vector_t *v, size_t initial_buf_siz, size_t member_size)`
- `gerror_t vector_destroy (vector_t *v)`
- `size_t vector_get_min_buf_siz (void)`
- `void vector_set_min_buf_siz (size_t new_min_buf_siz)`
- `gerror_t vector_resize_buffer (vector_t *v, size_t n_elements)`
- `gerror_t vector_at (vector_t *v, size_t index, void *elem)`
- `gerror_t vector_set_elem_at (vector_t *v, size_t index, void *elem)`
- `gerror_t vector_add (vector_t *v, void *elem)`
- `void * vector_ptr_at (vector_t *v, size_t index)`
- `gerror_t vector_append (vector_t *v0, vector_t *v1)`
- `gerror_t vector_append_buffer (vector_t *v, void *buffer, size_t size)`

Variables

- `size_t vector_min_siz = VECTOR_MIN_SIZ`

4.14.1 Macro Definition Documentation

4.14.1.1 VECTOR_MIN_SIZ

```
#define VECTOR_MIN_SIZ 8
```

4.14.2 Function Documentation

4.14.2.1 vector_add()

```
gerror_t vector_add (
    vector_t * v,
    void * elem )
```

adds the `elem` in the structure `vector_t` pointed by `v`.

Parameters

<i>v</i>	a pointer to <code>vector_t</code>
<i>elem</i>	the element to be add in <i>v</i>

Returns

`GERROR_OK` in case of success operation; `GERROR_NULL_STRUCTURE` in case *v* is a NULL pointer

4.14.2.2 vector_append()

```
gerror_t vector_append (
    vector_t * v0,
    vector_t * v1 )
```

Appends the vector `v1` to the vector `v0`.

Parameters

<i>v0</i>	vector structure
<i>v1</i>	vector structure to be appended

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCTURE in case *v* is a NULL

4.14.2.3 vector_append_buffer()

```
gerror_t vector_append_buffer (
    vector_t * v,
    void * buffer,
    size_t size )
```

Appends *size* members pointed by *buffer* in *v*.

Parameters

<i>v</i>	a pointer to <code>vector_t</code> structure
<i>buffer</i>	pointer to the elements with the same size to <i>v->member_size</i> to be appended
<i>size</i>	number of elements pointed by the <i>buffer</i>

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCTURE in case *v* is a NULL

4.14.2.4 vector_at()

```
gerror_t vector_at (
    vector_t * v,
    size_t index,
    void * elem )
```

Get the element in the *index* position indexed by the `vector_t` structure pointed by *v*.

Parameters

<i>v</i>	a pointer to <code>vector_t</code>
<i>index</i>	index of the position
<i>elem</i>	pointer to a previous allocated memory that will receive the element

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case *v* is a NULL pointer

4.14.2.5 vector_create()

```
gerror_t vector_create (
    vector_t * v,
    size_t initial_buf_siz,
    size_t member_size )
```

Populate the `vector_t` structure pointed by `v` and allocates `member_size*initial_size` for initial buffer↵
_size.

Parameters

<code>v</code>	a pointer to <code>vector_t</code> structure already allocated;
<code>initial_buf_size</code>	number of the members of the initial allocated buffer;
<code>member_size</code>	size of every member indexed by <code>v</code> .

Returns

`GERROR_OK` in case of success operation; `GERROR_NULL_STRUCURE` in case `v` is a NULL pointer

4.14.2.6 vector_destroy()

```
gerror_t vector_destroy (
    vector_t * v )
```

Destroy the structure `vector_t` pointed by `v`.

Parameters

<code>v</code>	a pointer to <code>vector_t</code> structure
----------------	--

Returns

`GERROR_OK` in case of success operation; `GERROR_NULL_STRUCURE` in case `v` is a NULL pointer

4.14.2.7 vector_get_min_buf_siz()

```
size_t vector_get_min_buf_siz (
    void )
```

Returns the `vector_min_siz`: a private variable that holds the minimal number of elements that `vector_t` will index. This variable is important for avoid multiple small resizes in the `vector_t` container.

Returns

`vector_min_siz`

4.14.2.8 `vector_ptr_at()`

```
void* vector_ptr_at (
    vector_t * v,
    size_t index )
```

Calculate the pointer at `index` position.

Parameters

<code>v</code>	a pointer to <code>vector_t</code>
<code>index</code>	index of the pointer

Returns

a pointer to the `index` element NULL in case of out of bound

4.14.2.9 `vector_resize_buffer()`

```
gerror_t vector_resize_buffer (
    vector_t * v,
    size_t n_elements )
```

Resize the buffer in the `vector_t` structure pointed by `v`.

Parameters

<code>v</code>	a pointer to <code>vector_t</code> structure.
<code>new_size</code>	the new size of the <code>v</code>

4.14.2.10 `vector_set_elem_at()`

```
gerror_t vector_set_elem_at (
    vector_t * v,
    size_t index,
    void * elem )
```

set the element at `index` pointed by `v` with the element pointed by `elem`.

Parameters

<code>v</code>	a pointer to <code>vector_t</code>
<code>index</code>	index of the position
<code>elem</code>	the element to be set in <code>v</code>

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `v` is a NULL pointer

4.14.2.11 `vector_set_min_buf_siz()`

```
void vector_set_min_buf_siz (
    size_t new_min_buf_siz )
```

Set the `vector_min_siz`: a private variable that holds the minimal number of elements that `vector_t` will index. This variable is important for avoid multiple small resizes in the `vector_t` container.

Parameters

<code>new_min_buf_siz</code>	the new size of <code>vector_min_siz</code>
------------------------------	---

4.14.3 Variable Documentation

4.14.3.1 `vector_min_siz`

```
size_t vector_min_siz = VECTOR_MIN_SIZ
```


Index

- adj
 - graph_t, 6
- buffer_size
 - vector_t, 13
- children
 - tnode_t, 12
- compare
 - priority_queue_t, 7
- compare_argument
 - priority_queue_t, 7
- compare_function
 - priority_queue.h, 21
- data
 - qnode_t, 8
 - snode_t, 10
 - vector_t, 13
- default_compare_function
 - priority_queue.c, 46
- E
 - graph_t, 6
- gerror.c
 - gerror_to_str, 42
 - gerror_to_string, 42
- gerror.h
 - gerror_t, 16
 - gerror_to_str, 16
- gerror_t
 - gerror.h, 16
- gerror_to_str
 - gerror.c, 42
 - gerror.h, 16
- gerror_to_string
 - gerror.c, 42
- graph.c
 - graph_add_edge, 43
 - graph_create, 43
 - graph_destroy, 43
 - graph_get_label_at, 44
 - graph_set_label_at, 44
- graph.h
 - graph_add_edge, 18
 - graph_create, 18
 - graph_destroy, 19
 - graph_get_label_at, 19
 - graph_set_label_at, 19
 - graph_t, 18
- graph_add_edge
 - graph.c, 43
 - graph.h, 18
- graph_create
 - graph.c, 43
 - graph.h, 18
- graph_destroy
 - graph.c, 43
 - graph.h, 19
- graph_get_label_at
 - graph.c, 44
 - graph.h, 19
- graph_set_label_at
 - graph.c, 44
 - graph.h, 19
- graph_t, 5
 - adj, 6
 - E, 6
 - graph.h, 18
 - label, 6
 - member_size, 6
 - V, 6
- head
 - queue_t, 9
 - stack_t, 11
- include/gerror.h, 15
- include/graph.h, 17
- include/priority_queue.h, 20
- include/queue.h, 25
- include/stack.h, 29
- include/trie.h, 32
- include/vector.h, 36
- LEFT
 - priority_queue.c, 46
- label
 - graph_t, 6
- max_heapify
 - priority_queue.c, 46
- member_size
 - graph_t, 6
 - priority_queue_t, 7
 - queue_t, 9
 - stack_t, 11
 - trie_t, 13
 - vector_t, 14
- NBYTE

- trie.h, 33
- next
 - qnode_t, 8
 - snode_t, 10
- node_at
 - trie.c, 54
- nswap
 - priority_queue.c, 46
- PARENT
 - priority_queue.c, 46
- pqueue_add
 - priority_queue.c, 46
 - priority_queue.h, 22
- pqueue_create
 - priority_queue.c, 47
 - priority_queue.h, 22
- pqueue_destroy
 - priority_queue.c, 47
 - priority_queue.h, 24
- pqueue_extract
 - priority_queue.c, 47
 - priority_queue.h, 24
- pqueue_max_priority
 - priority_queue.c, 48
 - priority_queue.h, 24
- pqueue_set_compare_function
 - priority_queue.c, 48
 - priority_queue.h, 25
- pqueue_t
 - priority_queue.h, 21
- prev
 - qnode_t, 8
 - snode_t, 10
- priority_queue.c
 - default_compare_function, 46
 - LEFT, 46
 - max_heapify, 46
 - nswap, 46
 - PARENT, 46
 - pqueue_add, 46
 - pqueue_create, 47
 - pqueue_destroy, 47
 - pqueue_extract, 47
 - pqueue_max_priority, 48
 - pqueue_set_compare_function, 48
 - RIGHT, 46
- priority_queue.h
 - compare_function, 21
 - pqueue_add, 22
 - pqueue_create, 22
 - pqueue_destroy, 24
 - pqueue_extract, 24
 - pqueue_max_priority, 24
 - pqueue_set_compare_function, 25
 - pqueue_t, 21
 - priority_queue_t, 22
 - queue_priority_t, 22
- priority_queue_t, 6
- compare, 7
- compare_argument, 7
- member_size, 7
- priority_queue.h, 22
- queue, 7
- size, 7
- qnode_t, 7
 - data, 8
 - next, 8
 - prev, 8
 - queue.h, 27
- queue
 - priority_queue_t, 7
- queue.c
 - queue_create, 50
 - queue_dequeue, 50
 - queue_destroy, 50
 - queue_enqueue, 51
 - queue_remove, 51
- queue.h
 - qnode_t, 27
 - queue_create, 27
 - queue_dequeue, 27
 - queue_destroy, 28
 - queue_enqueue, 28
 - queue_remove, 28
 - queue_t, 27
- queue_create
 - queue.c, 50
 - queue.h, 27
- queue_dequeue
 - queue.c, 50
 - queue.h, 27
- queue_destroy
 - queue.c, 50
 - queue.h, 28
- queue_enqueue
 - queue.c, 51
 - queue.h, 28
- queue_priority_t
 - priority_queue.h, 22
- queue_remove
 - queue.c, 51
 - queue.h, 28
- queue_t, 8
 - head, 9
 - member_size, 9
 - queue.h, 27
 - size, 9
 - tail, 9
- RIGHT
 - priority_queue.c, 46
- root
 - trie_t, 13
- size
 - priority_queue_t, 7

- queue_t, 9
- stack_t, 11
- trie_t, 13
- vector_t, 14
- snode_t, 9
 - data, 10
 - next, 10
 - prev, 10
 - stack.h, 30
- src/gerror.c, 41
- src/graph.c, 42
- src/priority_queue.c, 45
- src/queue.c, 49
- src/stack.c, 51
- src/trie.c, 54
- src/vector.c, 57
- stack.c
 - stack_create, 52
 - stack_destroy, 52
 - stack_pop, 53
 - stack_push, 53
- stack.h
 - snode_t, 30
 - stack_create, 30
 - stack_destroy, 31
 - stack_pop, 31
 - stack_push, 31
 - stack_t, 30
- stack_create
 - stack.c, 52
 - stack.h, 30
- stack_destroy
 - stack.c, 52
 - stack.h, 31
- stack_pop
 - stack.c, 53
 - stack.h, 31
- stack_push
 - stack.c, 53
 - stack.h, 31
- stack_t, 10
 - head, 11
 - member_size, 11
 - size, 11
 - stack.h, 30
- tail
 - queue_t, 9
- tnode_t, 11
 - children, 12
 - trie.h, 33
 - value, 12
- trie.c
 - node_at, 54
 - trie_add_element, 54
 - trie_create, 55
 - trie_destroy, 55
 - trie_destroy_tnode, 55
 - trie_get_element, 55
 - trie_get_node_or_allocate, 56
 - trie_remove_element, 56
 - trie_set_element, 56
- trie.h
 - NBYTE, 33
 - tnode_t, 33
 - trie_add_element, 33
 - trie_create, 34
 - trie_destroy, 34
 - trie_get_element, 34
 - trie_get_node_or_allocate, 35
 - trie_remove_element, 35
 - trie_set_element, 35
 - trie_t, 33
- trie_add_element
 - trie.c, 54
 - trie.h, 33
- trie_create
 - trie.c, 55
 - trie.h, 34
- trie_destroy
 - trie.c, 55
 - trie.h, 34
- trie_destroy_tnode
 - trie.c, 55
- trie_get_element
 - trie.c, 55
 - trie.h, 34
- trie_get_node_or_allocate
 - trie.c, 56
 - trie.h, 35
- trie_remove_element
 - trie.c, 56
 - trie.h, 35
- trie_set_element
 - trie.c, 56
 - trie.h, 35
- trie_t, 12
 - member_size, 13
 - root, 13
 - size, 13
 - trie.h, 33
- V
 - graph_t, 6
- VECTOR_MIN_SIZ
 - vector.c, 58
- value
 - tnode_t, 12
- vector.c
 - VECTOR_MIN_SIZ, 58
 - vector_add, 58
 - vector_append, 58
 - vector_append_buffer, 60
 - vector_at, 60
 - vector_create, 60
 - vector_destroy, 61
 - vector_get_min_buf_siz, 61
 - vector_min_siz, 63

- vector_ptr_at, [61](#)
- vector_resize_buffer, [62](#)
- vector_set_elem_at, [62](#)
- vector_set_min_buf_siz, [63](#)
- vector.h
 - vector_add, [37](#)
 - vector_append, [37](#)
 - vector_append_buffer, [38](#)
 - vector_at, [38](#)
 - vector_create, [38](#)
 - vector_destroy, [39](#)
 - vector_get_min_buf_siz, [39](#)
 - vector_ptr_at, [39](#)
 - vector_resize_buffer, [40](#)
 - vector_set_elem_at, [40](#)
 - vector_set_min_buf_siz, [41](#)
 - vector_t, [37](#)
- vector_add
 - vector.c, [58](#)
 - vector.h, [37](#)
- vector_append
 - vector.c, [58](#)
 - vector.h, [37](#)
- vector_append_buffer
 - vector.c, [60](#)
 - vector.h, [38](#)
- vector_at
 - vector.c, [60](#)
 - vector.h, [38](#)
- vector_create
 - vector.c, [60](#)
 - vector.h, [38](#)
- vector_destroy
 - vector.c, [61](#)
 - vector.h, [39](#)
- vector_get_min_buf_siz
 - vector.c, [61](#)
 - vector.h, [39](#)
- vector_min_siz
 - vector.c, [63](#)
- vector_ptr_at
 - vector.c, [61](#)
 - vector.h, [39](#)
- vector_resize_buffer
 - vector.c, [62](#)
 - vector.h, [40](#)
- vector_set_elem_at
 - vector.c, [62](#)
 - vector.h, [40](#)
- vector_set_min_buf_siz
 - vector.c, [63](#)
 - vector.h, [41](#)
- vector_t, [13](#)
 - buffer_size, [13](#)
 - data, [13](#)
 - member_size, [14](#)
 - size, [14](#)
 - vector.h, [37](#)