libgenerics

Generated by Doxygen 1.8.11

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 qnode_t Struct Reference 6

3.2.1 Detailed Description 6

3.2.2 Member Data Documentation 7

3.2.2.1 data 7

3.2.2.2 next 7

3.2.2.3 prev 7

3.3 queue_t Struct Reference 7

3.3.1 Detailed Description 7

3.3.2 Member Data Documentation 8

3.3.2.1 head . . . 8
3.3.2.2 member_size . . . 8
3.3.2.3 size . . . 8
3.3.2.4 tail . . . 8
3.4 snode_t Struct Reference . . . 8
3.4.1 Detailed Description . . . 8
3.4.2 Member Data Documentation . . . 8
3.4.2.1 data . . . 8
3.4.2.2 next . . . 8
3.4.2.3 prev . . . 8
3.5 stack_t Struct Reference . . . 9
3.5.1 Detailed Description . . . 9
3.5.2 Member Data Documentation . . . 9
3.5.2.1 head . . . 9
3.5.2.2 member_size . . . 9
3.5.2.3 size . . . 9
3.6 tnode_t Struct Reference . . . 10
3.6.1 Detailed Description . . . 10
3.6.2 Member Data Documentation . . . 10
3.6.2.1 children . . . 10
3.6.2.2 value . . . 10
3.7 trie_t Struct Reference . . . 10
3.7.1 Detailed Description . . . 11
3.7.2 Member Data Documentation . . . 11
3.7.2.1 member_size . . . 11
3.7.2.2 root . . . 11
3.7.2.3 size . . . 11
3.8 vector_t Struct Reference . . . 11
3.8.1 Member Data Documentation . . . 11
3.8.1.1 buffer_size . . . 11
3.8.1.2 data . . . 11
3.8.1.3 member_size . . . 11
3.8.1.4 size . . . 11

4 File Documentation 13

4.1 include/gerror.h File Reference . . . 13

4.1.1 Typedef Documentation . . . 14

4.1.1.1 gerror_t . . . 14

4.1.2 Enumeration Type Documentation . . . 14

4.1.2.1 gerror_t . . . 14

4.1.3 Function Documentation . . . 14

4.1.3.1 gerror_to_str(gerror_t g) . . . 14

4.2 include/graph.h File Reference . . . 14

4.2.1 Typedef Documentation . . . 16

4.2.1.1 graph_t . . . 16

4.2.2 Function Documentation . . . 16

4.2.2.1 graph_add_edge(graph_t *g, size_t from, size_t to) . . . 16

4.2.2.2 graph_create(graph_t *g, size_t size, size_t member_size) . . . 16

4.2.2.3 graph_destroy(graph_t *g) . . . 17

4.2.2.4 graph_get_label_at(graph_t *g, size_t index, void *label) . . . 17

4.2.2.5 graph_set_label_at(graph_t *g, size_t index, void *label) . . . 17

4.3 include/queue.h File Reference . . . 17

4.3.1 Typedef Documentation . . . 19

4.3.1.1 qnode_t . . . 19

4.3.1.2 queue_t . . . 19

4.3.2 Function Documentation . . . 19

4.3.2.1 queue_create(struct queue_t *q, size_t member_size) . . . 19

4.3.2.2 queue_dequeue(struct queue_t *q, void *e) . . . 19

4.3.2.3 queue_destroy(struct queue_t *q) . . . 20

4.3.2.4 queue_enqueue(struct queue_t *q, void *e) . . . 20

4.3.2.5 queue_remove(struct queue_t *q, struct qnode_t *node, void *e) . . . 20

4.4 include/stack.h File Reference . . . 21

4.4.1 Typedef Documentation . . . 22

4.4.1.1 snode_t . . . 22

4.4.1.2 stack_t . . . 22
4.4.2 Function Documentation . . . 22
4.4.2.1 stack_create(struct stack_t *q, size_t member_size) . . . 22
4.4.2.2 stack_destroy(struct stack_t *q) . . . 22
4.4.2.3 stack_pop(struct stack_t *q, void *e) . . . 23
4.4.2.4 stack_push(struct stack_t *q, void *e) . . . 23
4.5 include/trie.h File Reference . . . 23
4.5.1 Macro Definition Documentation . . . 25
4.5.1.1 NBYTE . . . 25
4.5.2 Typedef Documentation . . . 25
4.5.2.1 tnode_t . . . 25
4.5.2.2 trie_t . . . 25
4.5.3 Function Documentation . . . 25
4.5.3.1 trie_add_element(struct trie_t *t, void *string, size_t size, void *elem) . . . 25
4.5.3.2 trie_create(struct trie_t *t, size_t member_size) . . . 25
4.5.3.3 trie_destroy(struct trie_t *t) . . . 26
4.5.3.4 trie_get_element(struct trie_t *t, void *string, size_t size, void *elem) . . . 26
4.5.3.5 trie_get_node_or_allocate(struct trie_t *t, void *string, size_t size) . . . 26
4.5.3.6 trie_remove_element(struct trie_t *t, void *string, size_t size) . . . 26
4.5.3.7 trie_set_element(struct trie_t *t, void *string, size_t size, void *elem) . . . 26
4.6 include/vector.h File Reference . . . 27
4.6.1 Typedef Documentation . . . 28
4.6.1.1 vector_t . . . 28
4.6.2 Function Documentation . . . 28
4.6.2.1 vector_add(vector_t *v, void *elem) . . . 28
4.6.2.2 vector_at(vector_t *v, size_t index, void *elem) . . . 28
4.6.2.3 vector_create(vector_t *v, size_t initial_size, size_t member_size) . . . 29
4.6.2.4 vector_destroy(vector_t *v) . . . 29
4.6.2.5 vector_get_min_buf_siz(void) . . . 29
4.6.2.6 vector_resize_buffer(vector_t *v, size_t new_size) . . . 29

4.6.2.7 vector_set_elem_at(vector_t *v, size_t index, void *elem) . . . 30
4.6.2.8 vector_set_min_buf_siz(size_t new_min_buf_size) . . . 30
4.7 src/gerror.c File Reference . . . 31
4.7.1 Function Documentation . . . 31
4.7.1.1 gerror_to_str(gerror_t g) . . . 31
4.7.2 Variable Documentation . . . 31
4.7.2.1 gerror_to_string . . . 31
4.8 src/graph.c File Reference . . . 32
4.8.1 Function Documentation . . . 32
4.8.1.1 graph_add_edge(graph_t *g, size_t from, size_t to) . . . 32
4.8.1.2 graph_create(graph_t *g, size_t size, size_t member_size) . . . 33
4.8.1.3 graph_destroy(graph_t *g) . . . 33
4.8.1.4 graph_get_label_at(graph_t *g, size_t index, void *label) . . . 33
4.8.1.5 graph_set_label_at(graph_t *g, size_t index, void *label) . . . 34
4.9 src/queue.c File Reference . . . 34
4.9.1 Function Documentation . . . 35
4.9.1.1 queue_create(struct queue_t *q, size_t member_size) . . . 35
4.9.1.2 queue_dequeue(struct queue_t *q, void *e) . . . 35
4.9.1.3 queue_destroy(struct queue_t *q) . . . 35
4.9.1.4 queue_enqueue(struct queue_t *q, void *e) . . . 35
4.9.1.5 queue_remove(struct queue_t *q, struct qnode_t *node, void *e) . . . 36
4.10 src/stack.c File Reference . . . 36
4.10.1 Function Documentation . . . 37
4.10.1.1 stack_create(struct stack_t *s, size_t member_size) . . . 37
4.10.1.2 stack_destroy(struct stack_t *s) . . . 37
4.10.1.3 stack_pop(struct stack_t *s, void *e) . . . 38
4.10.1.4 stack_push(struct stack_t *s, void *e) . . . 38
4.11 src/trie.c File Reference . . . 38
4.11.1 Function Documentation . . . 39
4.11.1.1 node_at(struct trie_t *t, void *string, size_t size) . . . 39

4.11.1.2 trie_add_element(struct trie_t *t, void *string, size_t size, void *elem) 39
4.11.1.3 trie_create(struct trie_t *t, size_t member_size) 40
4.11.1.4 trie_destroy(struct trie_t *t) 40
4.11.1.5 trie_destroy_tnode(struct tnode_t *node) 40
4.11.1.6 trie_get_element(struct trie_t *t, void *string, size_t size, void *elem) 40
4.11.1.7 trie_get_node_or_allocate(struct trie_t *t, void *string, size_t size) 40
4.11.1.8 trie_remove_element(struct trie_t *t, void *string, size_t size) 40
4.11.1.9 trie_set_element(struct trie_t *t, void *string, size_t size, void *elem) 41
4.12 src/vector.c File Reference 41
4.12.1 Macro Definition Documentation 42
4.12.1.1 VECTOR_MIN_SIZ 42
4.12.2 Function Documentation 42
4.12.2.1 vector_add(vector_t *v, void *elem) 42
4.12.2.2 vector_at(vector_t *v, size_t index, void *elem) 42
4.12.2.3 vector_create(vector_t *v, size_t initial_buf_siz, size_t member_size) 43
4.12.2.4 vector_destroy(vector_t *v) 43
4.12.2.5 vector_get_min_buf_siz(void) 43
4.12.2.6 vector_resize_buffer(vector_t *v, size_t n_elements) 43
4.12.2.7 vector_set_elem_at(vector_t *v, size_t index, void *elem) 44
4.12.2.8 vector_set_min_buf_siz(size_t new_min_buf_siz) 44
4.12.3 Variable Documentation 44
4.12.3.1 vector_min_siz 44

Index **45**

Chapter 1

Class Index

1.1 Class List

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

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

include/gerror.h . . . 13
include/graph.h . . . 14
include/queue.h . . . 17
include/stack.h . . . 21
include/trie.h . . . 23
include/vector.h . . . 27
src/gerror.c . . . 31
src/graph.c . . . 32
src/queue.c . . . 34
src/stack.c . . . 36
src/trie.c . . . 38
src/vector.c . . . 41

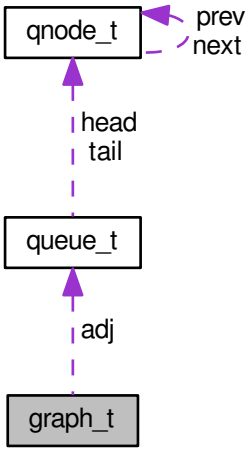
Chapter 3

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 struct queue_t* graph_t::adj

3.1.2.2 size_t graph_t::E

3.1.2.3 void* graph_t::label

3.1.2.4 size_t graph_t::member_size

3.1.2.5 size_t graph_t::V

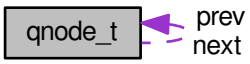
The documentation for this struct was generated from the following file:

- include/graph.h

3.2 qnode_t Struct Reference

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

Collaboration diagram for qnode_t:



Public Attributes

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

3.2.1 Detailed Description

queue node.

3.2.2 Member Data Documentation

3.2.2.1 void* qnode_t::data

3.2.2.2 struct qnode_t* qnode_t::next

3.2.2.3 struct qnode_t* qnode_t::prev

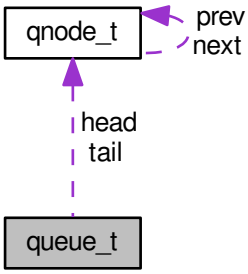
The documentation for this struct was generated from the following file:

- include/queue.h

3.3 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

3.3.1 Detailed Description

Represents a queue structure.

3.3.2 Member Data Documentation

3.3.2.1 struct qnode_t* queue_t::head

3.3.2.2 size_t queue_t::member_size

3.3.2.3 size_t queue_t::size

3.3.2.4 struct qnode_t* queue_t::tail

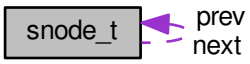
The documentation for this struct was generated from the following file:

- include/queue.h

3.4 snode_t Struct Reference

`#include <stack.h>`

Collaboration diagram for snode_t:



Public Attributes

- struct snode_t * next
- struct snode_t * prev
- void * data

3.4.1 Detailed Description

node of a stack

3.4.2 Member Data Documentation

3.4.2.1 void* snode_t::data

3.4.2.2 struct snode_t* snode_t::next

3.4.2.3 struct snode_t* snode_t::prev

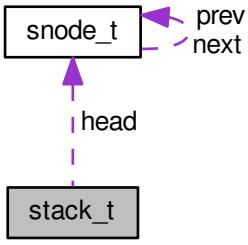
The documentation for this struct was generated from the following file:

- include/stack.h

3.5 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

3.5.1 Detailed Description

represents the stack structure.

3.5.2 Member Data Documentation

3.5.2.1 struct snode_t* stack_t::head

3.5.2.2 size_t stack_t::member_size

3.5.2.3 size_t stack_t::size

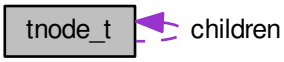
The documentation for this struct was generated from the following file:

- include/stack.h

3.6 tnode_t Struct Reference

`#include <trie.h>`

Collaboration diagram for tnode_t:



Public Attributes

- void * value
- struct tnode_t * children [NBYTE]

3.6.1 Detailed Description

node of a trie_t element.

3.6.2 Member Data Documentation

3.6.2.1 struct tnode_t* tnode_t::children[NBYTE]

3.6.2.2 void* tnode_t::value

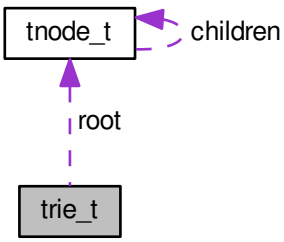
The documentation for this struct was generated from the following file:

- include/trie.h

3.7 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

3.7.1 Detailed Description

Represents the trie structure.

3.7.2 Member Data Documentation

3.7.2.1 size_t trie_t::member_size

3.7.2.2 struct tnode_t trie_t::root

3.7.2.3 size_t trie_t::size

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

- include/trie.h

3.8 vector_t Struct Reference

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

Public Attributes

- void * data
- size_t size
- size_t buffer_size
- size_t member_size

3.8.1 Member Data Documentation

3.8.1.1 size_t vector_t::buffer_size

3.8.1.2 void* vector_t::data

3.8.1.3 size_t vector_t::member_size

3.8.1.4 size_t vector_t::size

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

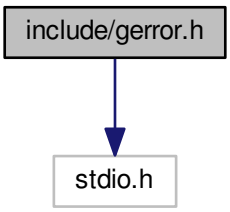
- include/vector.h

Chapter 4

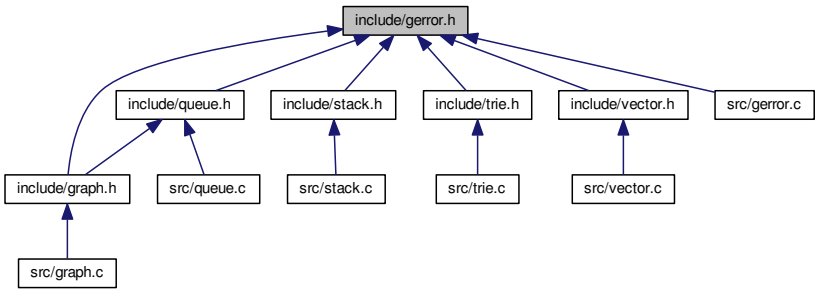
File Documentation

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_TRY_REMOVE_EMPTY_STRUCTURE, GERROR_TRY_ADD_EDGE_NO_VERTEX, GERR↩
  OR_ACCESS_OUT_OF_BOUND, GERROR_N_ERROR }

Functions

- char * gerror_to_str (gerror_t g)

4.1.1 Typedef Documentation

4.1.1.1 typedef enum gerror_t gerror_t

4.1.2 Enumeration Type Documentation

4.1.2.1 enum gerror_t

Enumerator

GERROR_OK

GERROR_NULL_STRUCTURE

GERROR_NULL_HEAD

GERROR_NULL_NODE

GERROR_TRY_REMOVE_EMPTY_STRUCTURE

GERROR_TRY_ADD_EDGE_NO_VERTEX

GERROR_ACCESS_OUT_OF_BOUND

GERROR_N_ERROR

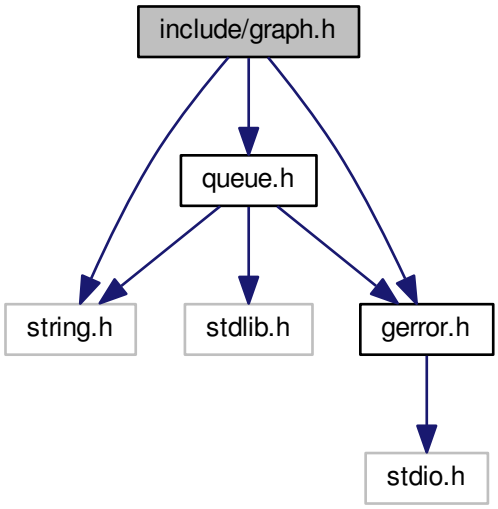
4.1.3 Function Documentation

4.1.3.1 char* gerror_to_str (gerror_t *g*)

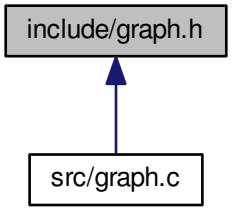
4.2 include/graph.h File Reference

```
#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 typedef struct graph_t graph_t

Graph structure and elements.

4.2.2 Function Documentation

4.2.2.1 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

g	pointer to a graph structure;
from	index of the first vertex;
to	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

4.2.2.2 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

g	pointer to a graph structure;
member_size	size of the elements that will be indexed by `g`

Returns

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

4.2.2.3 gerror_t graph_destroy (graph_t * g)

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

Parameters

g	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 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

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

Returns

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

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

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

Parameters

g	pointer to a graph structure;
index	index of the vertex;
label	the new label of the vertex positioned in `index`

Returns

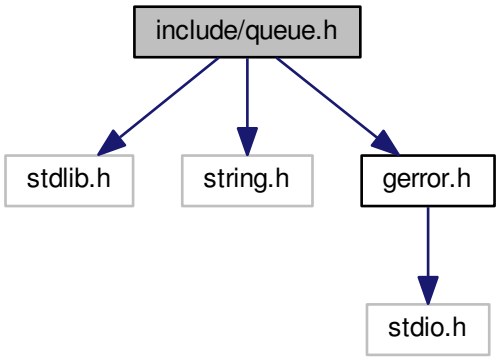
GERROR_OK in case of success operation; GERROR_ACCESS_OUT_OF_BOUND in case that `index` is out of bound

4.3 include/queue.h File Reference

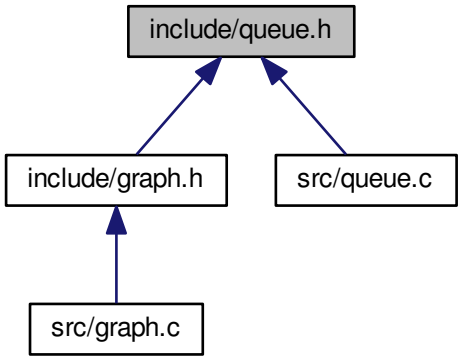
```
#include <stdlib.h>
```

```
#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)

4.3.1 Typedef Documentation

4.3.1.1 typedef struct qnode_t qnode_t

queue node.

4.3.1.2 typedef struct queue_t queue_t

Represents a queue structure.

4.3.2 Function Documentation

4.3.2.1 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

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

Returns

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

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

Dequeues the first element of the queue q

Parameters

q	pointer to a queue structure;
e	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.

4.3.2.3 gerror_t queue_destroy (struct queue_t * *q*)

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

Parameters

q	pointer to a queue structure;

Returns

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

4.3.2.4 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

4.3.2.5 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

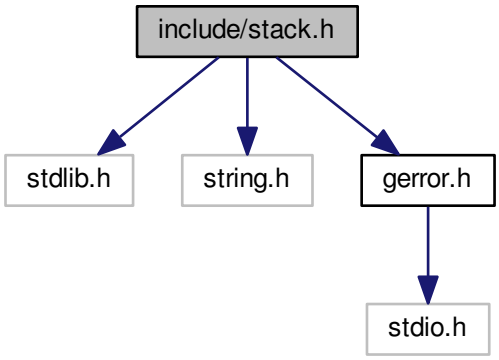
Returns

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.

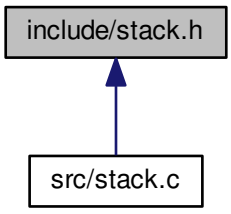
4.4 include/stack.h File Reference

```
#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)

4.4.1 Typedef Documentation

4.4.1.1 typedef struct snode_t snode_t

node of a stack

4.4.1.2 typedef struct stack_t stack_t

represents the stack structure.

4.4.2 Function Documentation

4.4.2.1 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

s	pointer to a stack structure;
member_size	size of the elements that will be indexed by `s`

Returns

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

4.4.2.2 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

s	pointer to a stack structure;

Returns

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

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

Pops the first element of the stack `s`.

Parameters

s	pointer to a stack structure;
e	pointer to the previous allocated element

Returns

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

4.4.2.4 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.

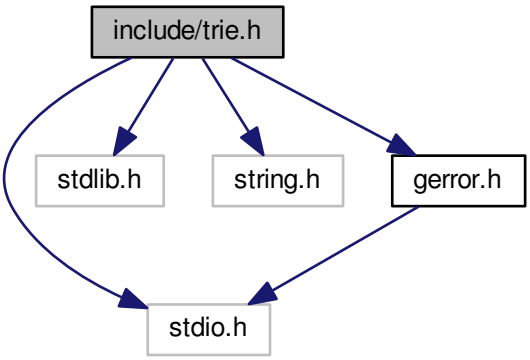
Returns

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

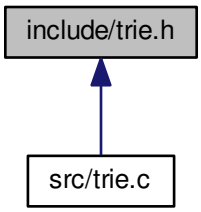
4.5 include/trie.h File Reference

```
#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)

4.5.1 Macro Definition Documentation

4.5.1.1 #define NBYTE (0x100)

4.5.2 Typedef Documentation

4.5.2.1 typedef struct tnode_t tnode_t

node of a trie_t element.

4.5.2.2 typedef struct trie_t trie_t

Represents the trie structure.

4.5.3 Function Documentation

4.5.3.1 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

t	pointer to the trie structure;
string	pointer to the string of bytes to map elem;
size	size of the string of bytes
elem	pointer to the element to add

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

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

Parameters

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

4.5.3.3 gerror_t trie_destroy (struct trie_t * *t*)

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

Parameters

t	pointer to the structure

4.5.3.4 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

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

Returns

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

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

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

Removes the element mapped by `string`.

Parameters

t	pointer to the structure trie_t;
string	pointer to the string of bytes to map elem;
size	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`

4.5.3.7 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.

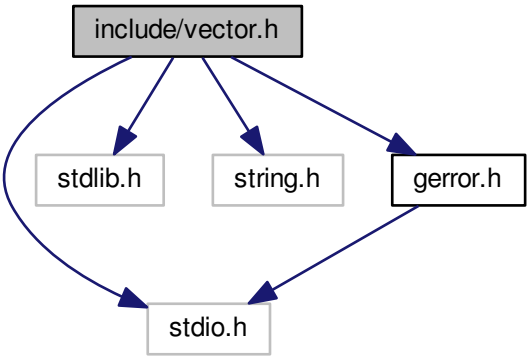
Parameters

t	pointer to the structure;
string	pointer to the string of bytes to map elem;
size	size of the string of bytes.
elem	pointer to the element to add

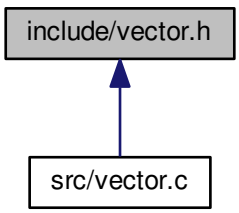
4.6 include/vector.h File Reference

```
#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_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)

4.6.1 Typedef Documentation

4.6.1.1 typedef struct vector_t vector_t

4.6.2 Function Documentation

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

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

Parameters

v	a pointer to `vector_t`
elem	the element to be add in `v`

Returns

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

4.6.2.2 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

v	a pointer to `vector_t`
index	index of the position
elem	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

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

Populate the vetor_t structure pointed by v and allocates member_size*initial_size for initial buffer_size.

Parameters

v	a pointer to vector_t structure already allocated;
inicial_buf_size	number of the members of the initial allocated buffer;
member_size	size of every member indexed by v.

Returns

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

4.6.2.4 gerror_t vector_destroy (vector_t * *v*)

Destroy the structure vector_t pointed by v.

Parameters

v	a pointer to vector_t structure

Returns

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

4.6.2.5 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

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

Resize the buffer in the vector_t strucuture pointed by v.

Parameters

v	a pointer to `vector_t` structure.
new_size	the new size of the `v`

4.6.2.7 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

v	a pointer to `vector_t`
index	index of the position
elem	the element to be set in `v`

Returns

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

4.6.2.8 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.

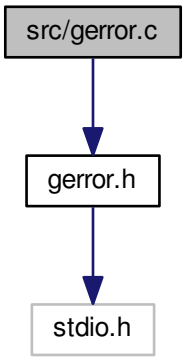
Parameters

new_min_buf_siz	the new size of `vector_min_siz`

4.7 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]

4.7.1 Function Documentation

4.7.1.1 char* gerror_to_str (gerror_t *g*)

4.7.2 Variable Documentation

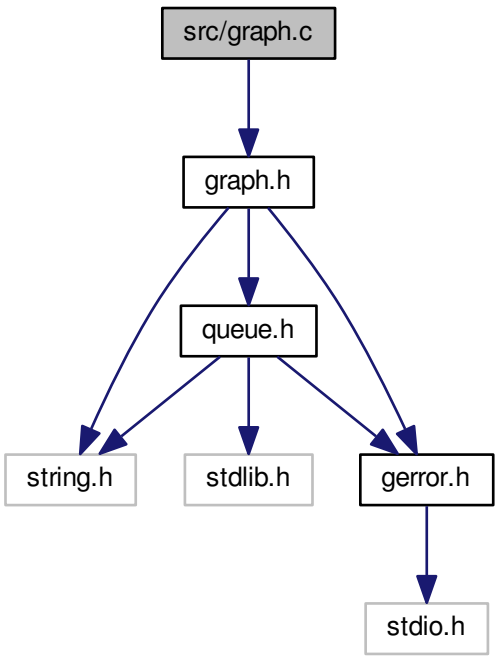
4.7.2.1 char* gerror_to_string[GERROR_N_ERROR]

Initial value:

```
= {
    "Success",
    "Null pointer to structure",
    "Null pointer to the head of structure",
    "Null pointer to the node",
    "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",
}
```

4.8 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)

4.8.1 Function Documentation

4.8.1.1 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

g	pointer to a graph structure;
from	index of the first vertex;
to	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

4.8.1.2 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

g	pointer to a graph structure;
member_size	size of the elements that will be indexed by `g`

Returns

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

4.8.1.3 gerror_t graph_destroy (graph_t * *g*)

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

Parameters

g	pointer to a graph structure;

Returns

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

4.8.1.4 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

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

Returns

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

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

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

Parameters

g	pointer to a graph structure;
index	index of the vertex;
label	the new label of the vertex positioned in `index`

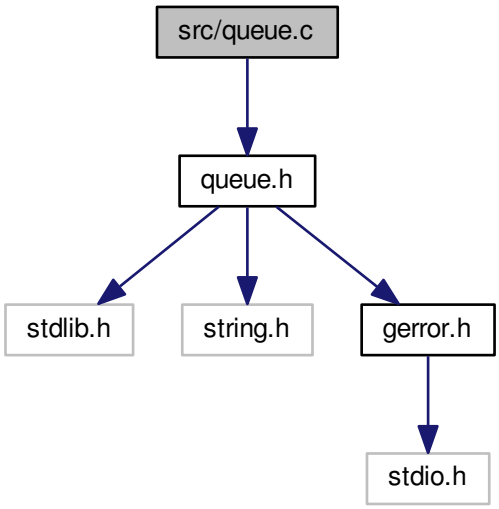
Returns

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

4.9 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)

4.9.1 Function Documentation

4.9.1.1 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

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

Returns

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

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

Dequeues the first element of the queue q

Parameters

q	pointer to a queue structure;
e	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_EMPT↩ Y_STRUCTURE in case that q has no element.

4.9.1.3 gerror_t queue_destroy (struct queue_t * q)

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

Parameters

q	pointer to a queue structure;

Returns

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

4.9.1.4 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

4.9.1.5 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

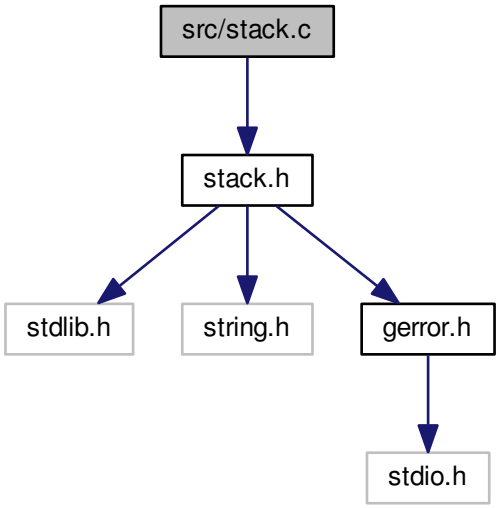
Returns

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.

4.10 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)

4.10.1 Function Documentation

4.10.1.1 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

s	pointer to a stack structure;
member_size	size of the elements that will be indexed by `s`

Returns

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

4.10.1.2 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

s	pointer to a stack structure;

Returns

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

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

Pops the first element of the stack `s`.

Parameters

s	pointer to a stack structure;
e	pointer to the previous allocated element

Returns

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

4.10.1.4 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.

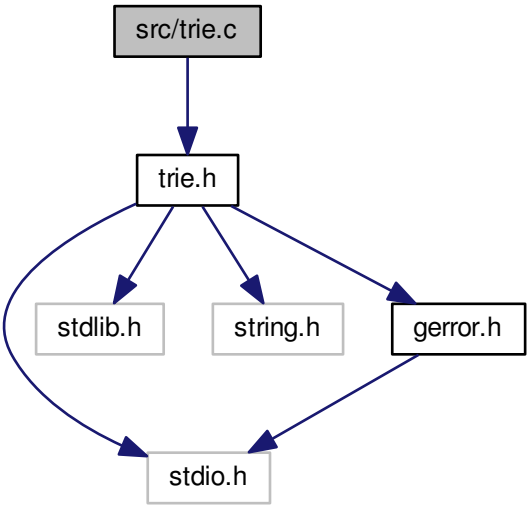
Returns

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

4.11 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)

4.11.1 Function Documentation

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

4.11.1.2 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

t	pointer to the trie structure;
string	pointer to the string of bytes to map elem;
size	size of the string of bytes
elem	pointer to the element to add

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

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

Parameters

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

4.11.1.4 gerror_t trie_destroy (struct trie_t * t)

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

Parameters

t	pointer to the structure

4.11.1.5 void trie_destroy_tnode (struct tnode_t * node)

4.11.1.6 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

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

Returns

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

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

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

Removes the element mapped by `string`.

Parameters

t	pointer to the structure trie_t;
string	pointer to the string of bytes to map elem;
size	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`

4.11.1.9 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.

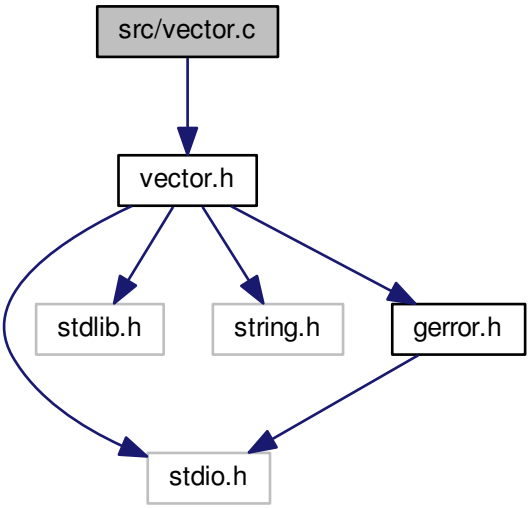
Parameters

t	pointer to the structure;
string	pointer to the string of bytes to map elem;
size	size of the string of bytes.
elem	pointer to the element to add

4.12 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)

Variables

- size_t vector_min_siz = VECTOR_MIN_SIZ

4.12.1 Macro Definition Documentation

4.12.1.1 #define VECTOR_MIN_SIZ 8

4.12.2 Function Documentation

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

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

Parameters

v	a pointer to `vector_t`
elem	the element to be add in `v`

Returns

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

4.12.2.2 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

v	a pointer to `vector_t`
index	index of the position
elem	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

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

Populate the vetor_t structure pointed by v and allocates member_size*initial_size for initial buffer_size.

Parameters

v	a pointer to vector_t structure already allocated;
inicial_buf_size	number of the members of the initial allocated buffer;
member_size	size of every member indexed by v.

Returns

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

4.12.2.4 gerror_t vector_destroy (vector_t * *v*)

Destroy the structure vector_t pointed by v.

Parameters

v	a pointer to vector_t structure

Returns

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

4.12.2.5 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

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

Resize the buffer in the vector_t strucuture pointed by v.

Parameters

v	a pointer to `vector_t` structure.
new_size	the new size of the `v`

4.12.2.7 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

v	a pointer to `vector_t`
index	index of the position
elem	the element to be set in `v`

Returns

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

4.12.2.8 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

new_min_buf_siz	the new size of `vector_min_siz`

4.12.3 Variable Documentation

4.12.3.1 size_t vector_min_siz = VECTOR_MIN_SIZ

Index

adj
    graph_t, 6

buffer_size
    vector_t, 11

children
    tnode_t, 10

data
    qnode_t, 7
    snode_t, 8
    vector_t, 11

E
    graph_t, 6

GERROR_ACCESS_OUT_OF_BOUND
    gerror.h, 14
GERROR_N_ERROR
    gerror.h, 14
GERROR_NULL_HEAD
    gerror.h, 14
GERROR_NULL_NODE
    gerror.h, 14
GERROR_NULL_STRUCTURE
    gerror.h, 14
GERROR_OK
    gerror.h, 14
GERROR_TRY_ADD_EDGE_NO_VERTEX
    gerror.h, 14
GERROR_TRY_REMOVE_EMPTY_STRUCTURE
    gerror.h, 14
gerror.c
    gerror_to_str, 31
    gerror_to_string, 31
gerror.h
    GERROR_ACCESS_OUT_OF_BOUND, 14
    GERROR_N_ERROR, 14
    GERROR_NULL_HEAD, 14
    GERROR_NULL_NODE, 14
    GERROR_NULL_STRUCTURE, 14
    GERROR_OK, 14
    GERROR_TRY_ADD_EDGE_NO_VERTEX, 14
    GERROR_TRY_REMOVE_EMPTY_STRUCTU↩
        RE, 14
    gerror_t, 14
    gerror_to_str, 14
gerror_t
    gerror.h, 14
gerror_to_str
    gerror.c, 31
    gerror.h, 14
gerror_to_string
    gerror.c, 31
graph.c
    graph_add_edge, 32
    graph_create, 33
    graph_destroy, 33
    graph_get_label_at, 33
    graph_set_label_at, 33
graph.h
    graph_add_edge, 16
    graph_create, 16
    graph_destroy, 16
    graph_get_label_at, 17
    graph_set_label_at, 17
    graph_t, 16
graph_add_edge
    graph.c, 32
    graph.h, 16
graph_create
    graph.c, 33
    graph.h, 16
graph_destroy
    graph.c, 33
    graph.h, 16
graph_get_label_at
    graph.c, 33
    graph.h, 17
graph_set_label_at
    graph.c, 33
    graph.h, 17
graph_t, 5
    adj, 6
    E, 6
    graph.h, 16
    label, 6
    member_size, 6
    V, 6

head
    queue_t, 8
    stack_t, 9

include/gerror.h, 13
include/graph.h, 14
include/queue.h, 17
include/stack.h, 21
include/trie.h, 23
include/vector.h, 27

label
graph_t, 6

member_size
graph_t, 6
queue_t, 8
stack_t, 9
trie_t, 11
vector_t, 11

NBYTE
trie.h, 25
next
qnode_t, 7
snode_t, 8
node_at
trie.c, 39

prev
qnode_t, 7
snode_t, 8

qnode_t, 6
data, 7
next, 7
prev, 7
queue.h, 19
queue.c
queue_create, 35
queue_dequeue, 35
queue_destroy, 35
queue_enqueue, 35
queue_remove, 36
queue.h
qnode_t, 19
queue_create, 19
queue_dequeue, 19
queue_destroy, 20
queue_enqueue, 20
queue_remove, 20
queue_t, 19
queue_create
queue.c, 35
queue.h, 19
queue_dequeue
queue.c, 35
queue.h, 19
queue_destroy
queue.c, 35
queue.h, 20
queue_enqueue
queue.c, 35
queue.h, 20
queue_remove
queue.c, 36
queue.h, 20
queue_t, 7
head, 8
member_size, 8
queue.h, 19
size, 8
tail, 8

root
trie_t, 11

size
queue_t, 8
stack_t, 9
trie_t, 11
vector_t, 11
snode_t, 8
data, 8
next, 8
prev, 8
stack.h, 22
src/gerror.c, 31
src/graph.c, 32
src/queue.c, 34
src/stack.c, 36
src/trie.c, 38
src/vector.c, 41
stack.c
stack_create, 37
stack_destroy, 37
stack_pop, 38
stack_push, 38
stack.h
snode_t, 22
stack_create, 22
stack_destroy, 22
stack_pop, 22
stack_push, 23
stack_t, 22
stack_create
stack.c, 37
stack.h, 22
stack_destroy
stack.c, 37
stack.h, 22
stack_pop
stack.c, 38
stack.h, 22
stack_push
stack.c, 38
stack.h, 23
stack_t, 9
head, 9
member_size, 9
size, 9
stack.h, 22

tail
queue_t, 8
tnode_t, 10
children, 10
trie.h, 25
value, 10

trie.c
node_at, 39
trie_add_element, 39
trie_create, 40
trie_destroy, 40
trie_destroy_tnode, 40
trie_get_element, 40
trie_get_node_or_allocate, 40
trie_remove_element, 40
trie_set_element, 41
trie.h
NBYTE, 25
tnode_t, 25
trie_add_element, 25
trie_create, 25
trie_destroy, 26
trie_get_element, 26
trie_get_node_or_allocate, 26
trie_remove_element, 26
trie_set_element, 26
trie_t, 25
trie_add_element
trie.c, 39
trie.h, 25
trie_create
trie.c, 40
trie.h, 25
trie_destroy
trie.c, 40
trie.h, 26
trie_destroy_tnode
trie.c, 40
trie_get_element
trie.c, 40
trie.h, 26
trie_get_node_or_allocate
trie.c, 40
trie.h, 26
trie_remove_element
trie.c, 40
trie.h, 26
trie_set_element
trie.c, 41
trie.h, 26
trie_t, 10
member_size, 11
root, 11
size, 11
trie.h, 25

V
graph_t, 6
VECTOR_MIN_SIZ
vector.c, 42
value
tnode_t, 10
vector.c
VECTOR_MIN_SIZ, 42
vector_add, 42
vector_at, 42
vector_create, 43
vector_destroy, 43
vector_get_min_buf_siz, 43
vector_min_siz, 44
vector_resize_buffer, 43
vector_set_elem_at, 44
vector_set_min_buf_siz, 44
vector.h
vector_add, 28
vector_at, 28
vector_create, 29
vector_destroy, 29
vector_get_min_buf_siz, 29
vector_resize_buffer, 29
vector_set_elem_at, 30
vector_set_min_buf_siz, 30
vector_t, 28
vector_add
vector.c, 42
vector.h, 28
vector_at
vector.c, 42
vector.h, 28
vector_create
vector.c, 43
vector.h, 29
vector_destroy
vector.c, 43
vector.h, 29
vector_get_min_buf_siz
vector.c, 43
vector.h, 29
vector_min_siz
vector.c, 44
vector_resize_buffer
vector.c, 43
vector.h, 29
vector_set_elem_at
vector.c, 44
vector.h, 30
vector_set_min_buf_siz
vector.c, 44
vector.h, 30
vector_t, 11
buffer_size, 11
data, 11
member_size, 11
size, 11
vector.h, 28