libgenerics

Generated by Doxygen 1.8.12

Contents

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

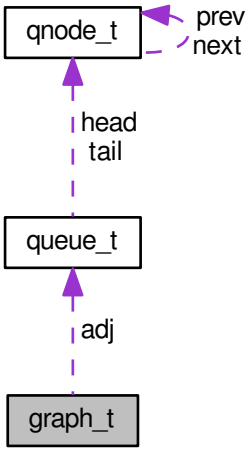
Chapter 3

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
```

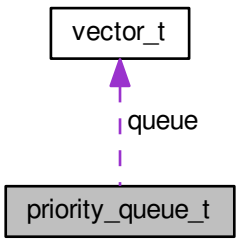
The documentation for this struct was generated from the following file:

- 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
```

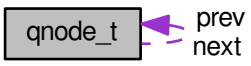
The documentation for this struct was generated from the following file:

- 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
```

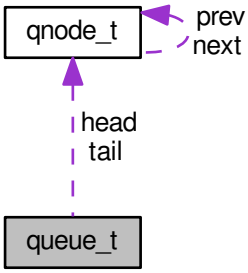
The documentation for this struct was generated from the following file:

- 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
```

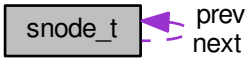
The documentation for this struct was generated from the following file:

- 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
```

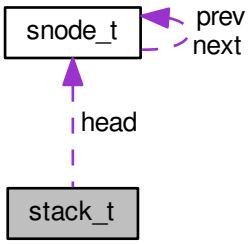
The documentation for this struct was generated from the following file:

- 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
```

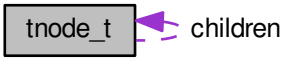
The documentation for this struct was generated from the following file:

- 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
```

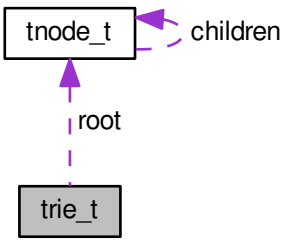
The documentation for this struct was generated from the following file:

- 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

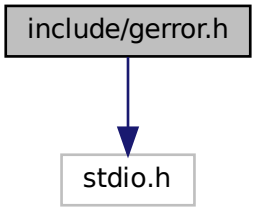
Chapter 4

File Documentation

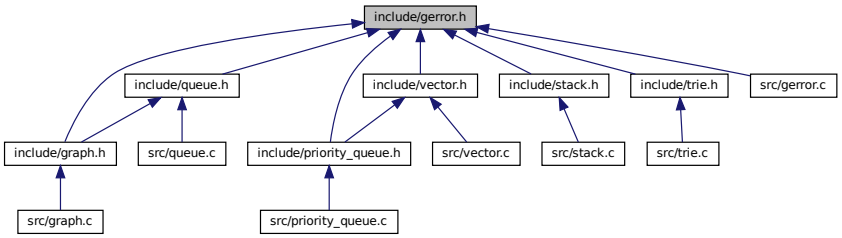
4.1 include/gerror.h File Reference

```
#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, GERRO↩
  R_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

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	

4.1.3 Function Documentation

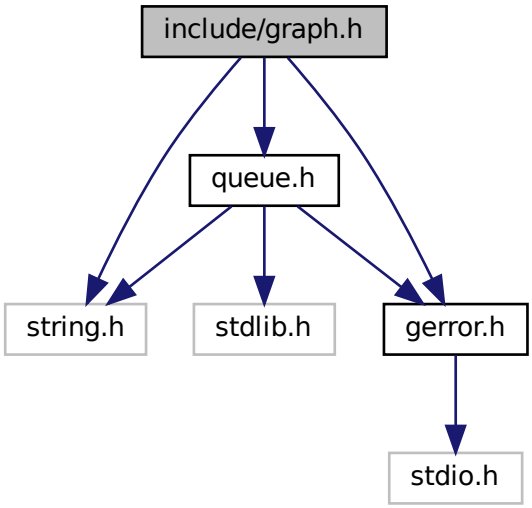
4.1.3.1 gerror_to_str()

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

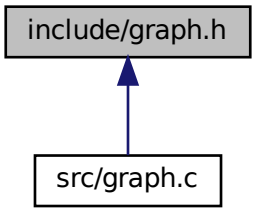
4.2 include/graph.h File Reference

```
#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

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 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

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 graph_destroy()

```
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 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

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 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

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

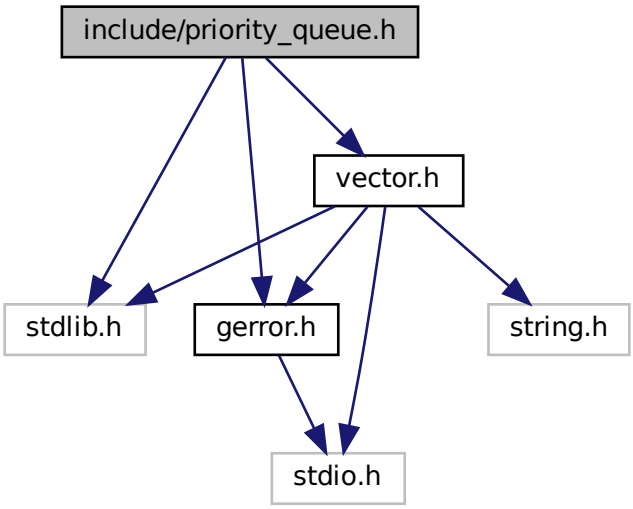
Returns

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

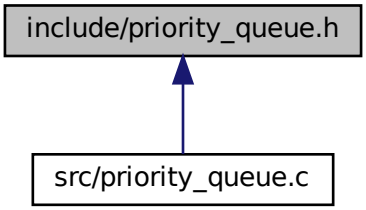
4.3 include/priority_queue.h File Reference

```
#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 datailed description of pqueue_add.

Parameters

p	previous allocated pqueue_t struct
e	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 inicialize 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

p	previous allocated pqueue_t struct
member_size	size in bytes of the indexed elements
function	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 datailed description of pqueue_destroy.

Parameters

p	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

p	previous allocated pqueue_t struct
e	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 datailed description of pqueue_↩ max_priority.

Parameters

p	previous allocated pqueue_t struct
e	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

p	previous allocated pqueue_t struct
function	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
argument	allocated pqueue_t struct

Returns

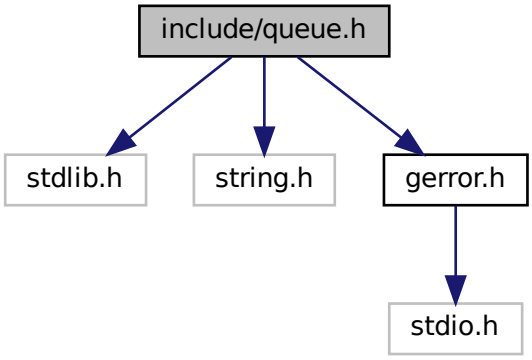
GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `t` is a NULL

4.4 include/queue.h File Reference

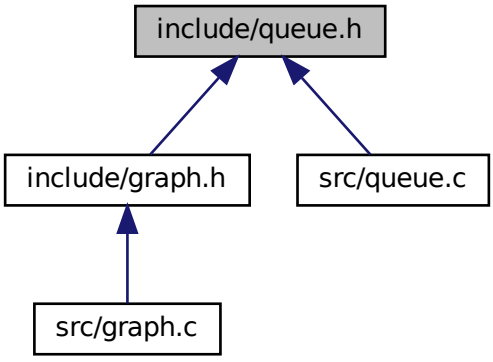
```
#include <stdlib.h>
#include <string.h>
```

```
#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

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

4.4.2.2 queue_dequeue()

```
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.

4.4.2.3 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

q	pointer to a queue structure;

Returns

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

4.4.2.4 queue_enqueue()

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

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

Parameters

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

4.4.2.5 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

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

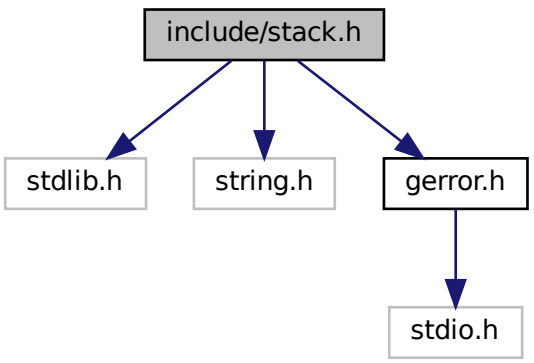
Returns

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.

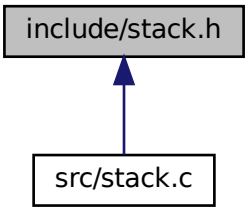
4.5 include/stack.h File Reference

```
#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

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.

4.5.2.2 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

s	pointer to a stack structure;

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE 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

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

4.5.2.4 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.

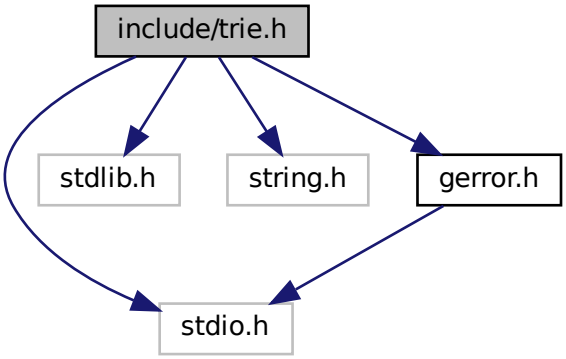
Returns

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

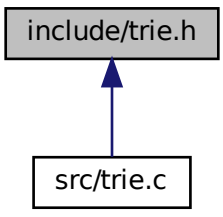
4.6 include/trie.h File Reference

```
#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

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

4.6.3.2 trie_create()

```
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.

4.6.3.3 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

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

4.6.3.5 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

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`

4.6.3.7 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.

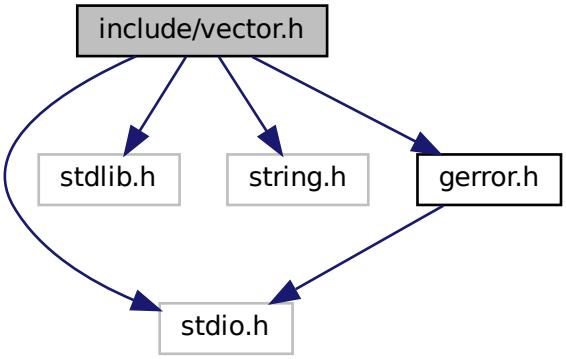
Parameters

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

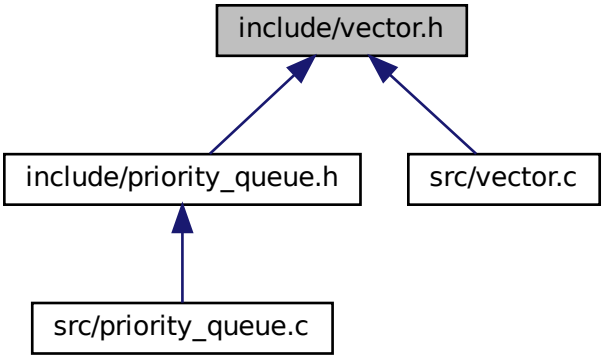
4.7 include/vector.h File Reference

```
#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

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

4.7.2.2 vector_append()

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

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

Parameters

v0	vector structure
v1	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

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

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

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

4.7.2.5 vector_create()

```
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

4.7.2.6 vector_destroy()

```
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

4.7.2.7 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

v	a pointer to `vector_t`
index	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` strucuture pointed by `v`.

Parameters

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

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

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

4.7.2.11 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.

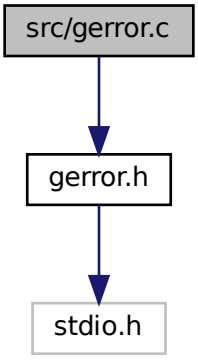
Parameters

new_min_buf_siz	the new size of `vector_min_siz`

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]
```

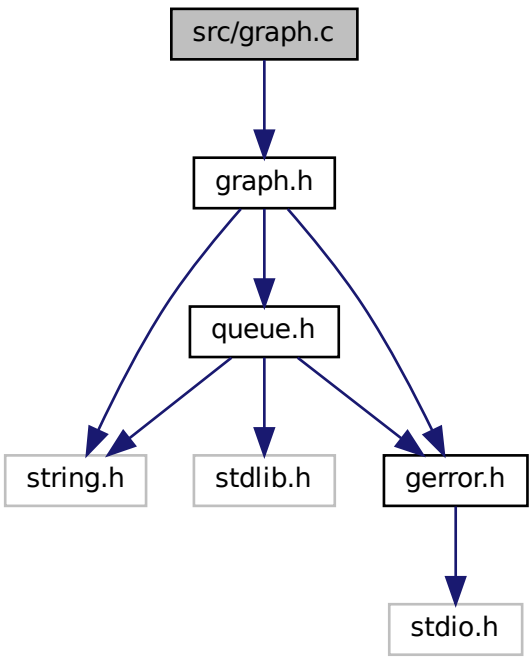
Initial value:

```
= {
    "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

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

4.9.1.2 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

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

4.9.1.3 graph_destroy()

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

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

Parameters

g	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

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

4.9.1.5 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

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

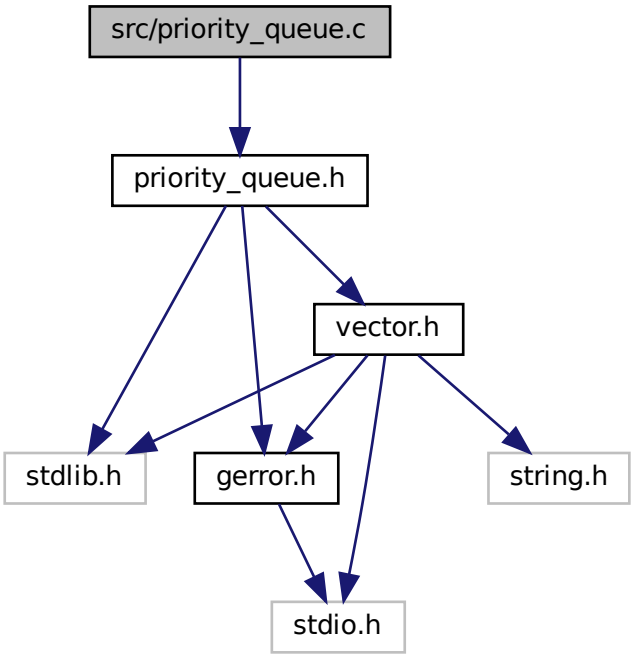
Returns

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

4.10 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 datailed description of pqueue_add.

Parameters

p	previous allocated pqueue_t struct
e	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 inicialize 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

p	previous allocated pqueue_t struct
member_size	size in bytes of the indexed elements
function	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

p	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

p	previous allocated pqueue_t struct
e	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 datailed description of pqueue_↩ max_priority.

Parameters

p	previous allocated pqueue_t struct
e	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

4.10.2.9 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

p	previous allocated pqueue_t struct
function	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
argument	allocated pqueue_t struct

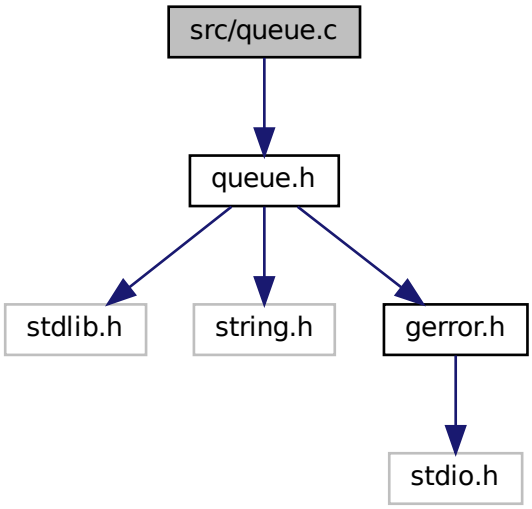
Returns

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

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

4.11.1.2 queue_dequeue()

```
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.

4.11.1.3 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

q	pointer to a queue structure;

Returns

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

4.11.1.4 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

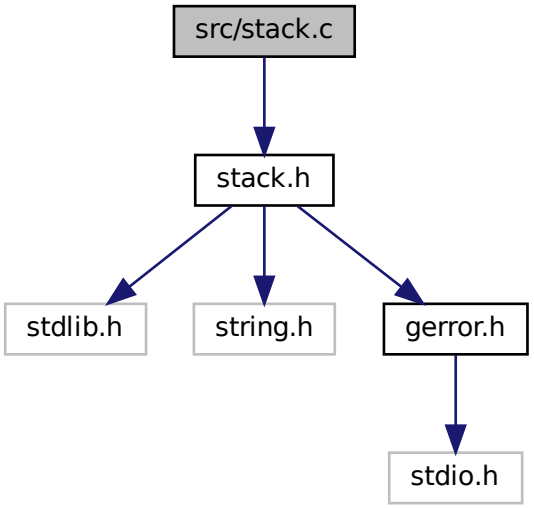
Returns

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

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.

4.12.1.2 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

s	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

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

4.12.1.4 stack_push()

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

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

Parameters

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

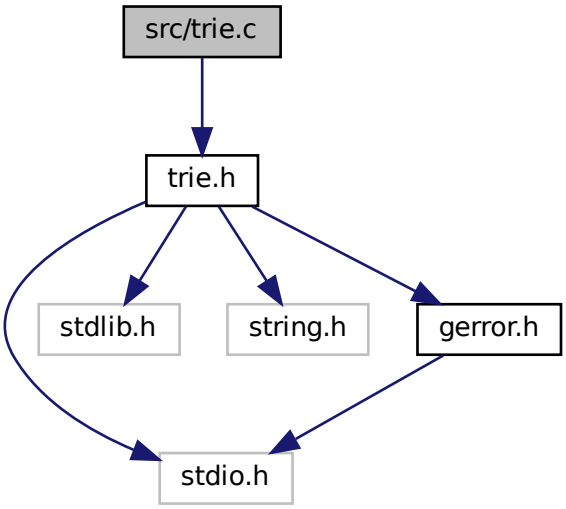
Returns

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

4.13 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

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

4.13.1.3 trie_create()

```
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.

4.13.1.4 trie_destroy()

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

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

Returns

GERROR_OK in case of success operation; GERROR_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

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

4.13.1.7 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

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`

4.13.1.9 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.

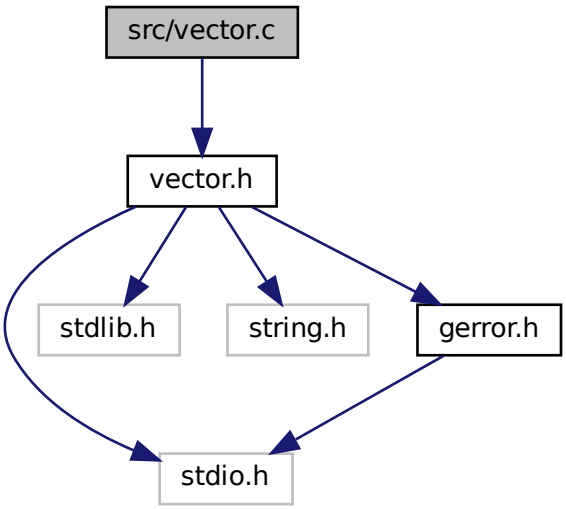
Parameters

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

4.14 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

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

4.14.2.2 vector_append()

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

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

Parameters

v0	vector structure
v1	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

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

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

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

4.14.2.5 vector_create()

```
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

4.14.2.6 vector_destroy()

```
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

4.14.2.7 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

v	a pointer to `vector_t`
index	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` strucuture pointed by `v`.

Parameters

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

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

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

4.14.2.11 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

new_min_buf_siz	the new size of `vector_min_siz`

4.14.3 Variable Documentation

4.14.3.1 vector_min_siz

```
size_t vector_min_siz = VECTOR_MIN_SIZ
```

Index