

Topologic

1.0.6

Generated by Doxygen 1.9.1



<b>1 Topologic</b>	<b>1</b>
1.1 Installation	1
1.1.1 Arch Linux	1
1.1.2 Manualy	1
1.2 TODO	2
1.3 Errata	2
1.3.1 Edge/vertex modification	2
1.3.2 Graph modification	2
1.3.3 Parameter Passing	2
1.3.4 Illegal Arguments	2
1.3.5 Graph Modification (cont.)	2
1.3.6 Data Structure Errata	3
1.3.7 SWITCH_UNSAFE	3
<b>2 Data Structure Index</b>	<b>5</b>
2.1 Data Structures	5
<b>3 File Index</b>	<b>7</b>
3.1 File List	7
<b>4 Data Structure Documentation</b>	<b>9</b>
4.1 AVLNode Struct Reference	9
4.1.1 Field Documentation	9
4.1.1.1 data	9
4.1.1.2 height	9
4.1.1.3 id	10
4.1.1.4 left	10
4.1.1.5 right	10
4.2 AVLTree Struct Reference	10
4.2.1 Field Documentation	10
4.2.1.1 root	11
4.2.1.2 size	11
4.3 destroy_edge_id_request Struct Reference	11
4.3.1 Field Documentation	11
4.3.1.1 a	11
4.3.1.2 id	11
4.4 destroy_edge_request Struct Reference	12
4.4.1 Field Documentation	12
4.4.1.1 a	12
4.4.1.2 b	12
4.5 destroy_vertex_id_request Struct Reference	12
4.5.1 Field Documentation	12
4.5.1.1 graph	13

4.5.1.2 id	13
4.6 destroy_vertex_request Struct Reference	13
4.6.1 Field Documentation	13
4.6.1.1 graph	13
4.6.1.2 vertex	13
4.7 edge Struct Reference	14
4.7.1 Detailed Description	14
4.7.2 Field Documentation	14
4.7.2.1 a	14
4.7.2.2 a_vars	14
4.7.2.3 b	14
4.7.2.4 b_vars	15
4.7.2.5 bi_edge	15
4.7.2.6 bi_edge_lock	15
4.7.2.7 edge_type	15
4.7.2.8 f	15
4.7.2.9 glbl	15
4.7.2.10 id	16
4.8 edge_request Struct Reference	16
4.8.1 Field Documentation	16
4.8.1.1 a	16
4.8.1.2 b	16
4.8.1.3 f	16
4.8.1.4 glbl	17
4.9 fireable Struct Reference	17
4.9.1 Field Documentation	17
4.9.1.1 args	17
4.9.1.2 color	17
4.9.1.3 graph	17
4.9.1.4 iloop	18
4.9.1.5 vertex	18
4.10 graph Struct Reference	18
4.10.1 Detailed Description	19
4.10.2 Field Documentation	19
4.10.2.1 black_fire	19
4.10.2.2 black_locked	19
4.10.2.3 black_vertex_count	19
4.10.2.4 color_lock	19
4.10.2.5 context	19
4.10.2.6 lock	20
4.10.2.7 lvl_verbose	20
4.10.2.8 max_loop	20

4.10.2.9 max_state_changes . . . . .	20
4.10.2.10 mem_option . . . . .	20
4.10.2.11 modify . . . . .	20
4.10.2.12 num_vertices . . . . .	21
4.10.2.13 pause . . . . .	21
4.10.2.14 pause_cond . . . . .	21
4.10.2.15 previous_color . . . . .	21
4.10.2.16 print_flag . . . . .	21
4.10.2.17 red_fire . . . . .	21
4.10.2.18 red_locked . . . . .	22
4.10.2.19 red_vertex_count . . . . .	22
4.10.2.20 remove_edges . . . . .	22
4.10.2.21 remove_vertices . . . . .	22
4.10.2.22 snapshot_timestamp . . . . .	22
4.10.2.23 start . . . . .	22
4.10.2.24 state . . . . .	23
4.10.2.25 state_count . . . . .	23
4.10.2.26 vertices . . . . .	23
4.11 mod_edge_vars_request Struct Reference . . . . .	23
4.11.1 Field Documentation . . . . .	23
4.11.1.1 edge_vars . . . . .	23
4.11.1.2 vertex . . . . .	24
4.12 mod_vertex_request Struct Reference . . . . .	24
4.12.1 Field Documentation . . . . .	24
4.12.1.1 f . . . . .	24
4.12.1.2 glbl . . . . .	24
4.12.1.3 vertex . . . . .	24
4.13 request Struct Reference . . . . .	25
4.13.1 Detailed Description . . . . .	25
4.13.2 Field Documentation . . . . .	25
4.13.2.1 args . . . . .	25
4.13.2.2 f . . . . .	25
4.13.2.3 request . . . . .	25
4.14 shared_edge Union Reference . . . . .	26
4.14.1 Field Documentation . . . . .	26
4.14.1.1 edge_data . . . . .	26
4.14.1.2 vertex_data . . . . .	26
4.15 stack Struct Reference . . . . .	26
4.15.1 Field Documentation . . . . .	26
4.15.1.1 length . . . . .	27
4.15.1.2 root . . . . .	27
4.16 stack_node Struct Reference . . . . .	27

4.16.1 Field Documentation	27
4.16.1.1 data	27
4.16.1.2 next	27
4.17 vertex Struct Reference	28
4.17.1 Detailed Description	28
4.17.2 Field Documentation	28
4.17.2.1 context	28
4.17.2.2 edge_tree	28
4.17.2.3 f	28
4.17.2.4 glbl	29
4.17.2.5 id	29
4.17.2.6 is_active	29
4.17.2.7 joining_vertices	29
4.17.2.8 lock	29
4.17.2.9 shared	29
4.18 vertex_request Struct Reference	30
4.18.1 Field Documentation	30
4.18.1.1 f	30
4.18.1.2 glbl	30
4.18.1.3 graph	30
4.18.1.4 id	30
4.19 vertex_result Struct Reference	31
4.19.1 Detailed Description	31
4.19.2 Field Documentation	31
4.19.2.1 edge_argv	31
4.19.2.2 edge_size	31
4.19.2.3 vertex_argv	31
4.19.2.4 vertex_size	31
<b>5 File Documentation</b>	<b>33</b>
5.1 include/AVL.h File Reference	33
5.1.1 Function Documentation	33
5.1.1.1 destroy_avl()	33
5.1.1.2 find()	34
5.1.1.3 init_avl()	34
5.1.1.4 inorder()	34
5.1.1.5 insert()	34
5.1.1.6 postorder()	35
5.1.1.7 preorder()	35
5.1.1.8 remove_ID()	35
5.1.1.9 stackify()	35
5.2 include/context.h File Reference	35

5.2.1 Enumeration Type Documentation	36
5.2.1.1 CONTEXT	36
5.3 include/edge.h File Reference	37
5.3.1 Enumeration Type Documentation	37
5.3.1.1 edge_type	37
5.4 include/graph.h File Reference	37
5.4.1 Macro Definition Documentation	38
5.4.1.1 MAX_ATTEMPTS	38
5.4.1.2 THREAD_ATTEMPT_SLEEP	38
5.4.2 Enumeration Type Documentation	38
5.4.2.1 MEM_OPTION	38
5.4.2.2 SNAPSHOT	39
5.4.2.3 STATES	39
5.4.2.4 VERBOSITY	39
5.5 include/header.h File Reference	40
5.6 include/request.h File Reference	40
5.6.1 Enumeration Type Documentation	40
5.6.1.1 REQUESTS	40
5.7 include/stack.h File Reference	41
5.7.1 Function Documentation	41
5.7.1.1 destroy_stack()	41
5.7.1.2 get()	42
5.7.1.3 init_stack()	42
5.7.1.4 pop()	42
5.7.1.5 push()	42
5.8 include/topologic.h File Reference	42
5.8.1 Macro Definition Documentation	44
5.8.1.1 _GNU_SOURCE	44
5.8.1.2 CREATE_BI_EDGE	44
5.8.1.3 CREATE_EDGE	44
5.8.1.4 CREATE_NULL_BI_EDGE	44
5.8.1.5 CREATE_REQUEST	45
5.8.1.6 CREATE_VERTEX	45
5.8.1.7 CREATE_VERTEX_GLBL	45
5.8.1.8 GRAPH_INIT	45
5.8.1.9 MAX_LOOPS	45
5.8.1.10 MODIFY_BI_EDGE	45
5.8.1.11 MODIFY_BI_EDGE_GLOBALS	46
5.8.1.12 MODIFY_EDGE	46
5.8.1.13 MODIFY_EDGE_GLOBALS	46
5.8.1.14 MODIFY_VERTEX	46
5.8.1.15 MODIFY_VERTEX_GLOBALS	46

5.8.1.16 PTHREAD_SLEEP_TIME . . . . .	46
5.8.1.17 TOPOLOGIC_DEBUG . . . . .	47
5.8.1.18 topologic_debug . . . . .	47
5.8.2 Function Documentation . . . . .	47
5.8.2.1 create_bi_edge() . . . . .	47
5.8.2.2 create_edge() . . . . .	47
5.8.2.3 create_request() . . . . .	48
5.8.2.4 create_vertex() . . . . .	48
5.8.2.5 destroy_graph() . . . . .	48
5.8.2.6 destroy_request() . . . . .	49
5.8.2.7 fire() . . . . .	49
5.8.2.8 fire_pthread() . . . . .	49
5.8.2.9 graph_init() . . . . .	50
5.8.2.10 modify_bi_edge() . . . . .	50
5.8.2.11 modify_edge() . . . . .	50
5.8.2.12 modify_shared_edge_vars() . . . . .	51
5.8.2.13 modify_vertex() . . . . .	51
5.8.2.14 parse_json() . . . . .	51
5.8.2.15 pause_graph() . . . . .	51
5.8.2.16 print_graph() . . . . .	52
5.8.2.17 process_requests() . . . . .	52
5.8.2.18 remove_bi_edge() . . . . .	52
5.8.2.19 remove_edge() . . . . .	52
5.8.2.20 remove_edge_id() . . . . .	53
5.8.2.21 remove_vertex() . . . . .	53
5.8.2.22 remove_vertex_id() . . . . .	53
5.8.2.23 resume_graph() . . . . .	54
5.8.2.24 run() . . . . .	54
5.8.2.25 start_set() . . . . .	54
5.8.2.26 submit_request() . . . . .	55
5.8.2.27 switch_vertex() . . . . .	55
5.9 include/vertex.h File Reference . . . . .	55
5.10 README.md File Reference . . . . .	56
5.11 src/AVL.c File Reference . . . . .	56
5.11.1 Function Documentation . . . . .	56
5.11.1.1 balance() . . . . .	56
5.11.1.2 create_node() . . . . .	57
5.11.1.3 destroy_avl() . . . . .	57
5.11.1.4 destroy_avl_nodes() . . . . .	57
5.11.1.5 find() . . . . .	57
5.11.1.6 find_node() . . . . .	58
5.11.1.7 init_avl() . . . . .	58



5.11.1.8 inorder()	58
5.11.1.9 inorder_nodes()	58
5.11.1.10 insert()	59
5.11.1.11 insert_node()	59
5.11.1.12 left_rotate()	59
5.11.1.13 max_height()	59
5.11.1.14 minNode()	60
5.11.1.15 postorder()	60
5.11.1.16 postorder_nodes()	60
5.11.1.17 preorder()	60
5.11.1.18 preorder_nodes()	60
5.11.1.19 remove_ID()	61
5.11.1.20 remove_node()	61
5.11.1.21 right_rotate()	61
5.11.1.22 stackify()	61
5.11.1.23 stackify_nodes()	62
5.12 src/edge.c File Reference	62
5.12.1 Function Documentation	62
5.12.1.1 create_bi_edge()	62
5.12.1.2 create_edge()	63
5.12.1.3 modify_bi_edge()	63
5.12.1.4 modify_edge()	63
5.12.1.5 remove_bi_edge()	64
5.12.1.6 remove_edge()	64
5.12.1.7 remove_edge_id()	64
5.13 src/graph.c File Reference	64
5.13.1 Function Documentation	65
5.13.1.1 destroy_graph()	65
5.13.1.2 destroy_graph_avl()	65
5.13.1.3 destroy_graph_stack()	65
5.13.1.4 graph_init()	66
5.14 src/request.c File Reference	66
5.14.1 Function Documentation	66
5.14.1.1 create_request()	66
5.14.1.2 destroy_request()	67
5.14.1.3 procces_request()	67
5.14.1.4 process_requests()	67
5.14.1.5 submit_request()	68
5.15 src/stack.c File Reference	68
5.15.1 Function Documentation	68
5.15.1.1 destroy_stack()	68
5.15.1.2 get()	68

5.15.1.3 <code>init_stack()</code> . . . . .	69
5.15.1.4 <code>pop()</code> . . . . .	69
5.15.1.5 <code>push()</code> . . . . .	69
5.16 <code>src/topologic.c</code> File Reference . . . . .	69
5.16.1 Function Documentation . . . . .	70
5.16.1.1 <code>fire()</code> . . . . .	70
5.16.1.2 <code>fire_pthread()</code> . . . . .	70
5.16.1.3 <code>pause_graph()</code> . . . . .	70
5.16.1.4 <code>resume_graph()</code> . . . . .	71
5.16.1.5 <code>run()</code> . . . . .	71
5.16.1.6 <code>run_single()</code> . . . . .	71
5.16.1.7 <code>sleep_ms()</code> . . . . .	71
5.16.1.8 <code>start_set()</code> . . . . .	72
5.16.1.9 <code>switch_vertex()</code> . . . . .	72
5.17 <code>src/topologic_json.c</code> File Reference . . . . .	72
5.17.1 Function Documentation . . . . .	72
5.17.1.1 <code>print_edges()</code> . . . . .	73
5.17.1.2 <code>print_graph()</code> . . . . .	73
5.17.1.3 <code>print_state()</code> . . . . .	73
5.18 <code>src/vertex.c</code> File Reference . . . . .	73
5.18.1 Function Documentation . . . . .	74
5.18.1.1 <code>create_vertex()</code> . . . . .	74
5.18.1.2 <code>modify_shared_edge_vars()</code> . . . . .	74
5.18.1.3 <code>modify_vertex()</code> . . . . .	75
5.18.1.4 <code>remove_vertex()</code> . . . . .	75
5.18.1.5 <code>remove_vertex_id()</code> . . . . .	75
<b>Index</b>	<b>77</b>

# Chapter 1

## Topologic

A library to simulate DFAs and Probabilistic DFAs using context switching. This library provides a way to build and define the behavior of a graph. The client is able to define the function each vertex and edge of the graph make and how the graph transitions from one (the first valid edge) or to many states, as well as which states the graph starts in.

The library provides a way for the client to define what information is saved locally to each vertex and edge, as well as which information the vertex and edges originating at that edge share. In addition the client can pass information to each initial state that will be passed and modifyable by vertices at each state transition.

This library can be used to create AI, ML, and simulations (such as of stock options, electron-electron interaction, etc.).

### 1.1 Installation

#### 1.1.1 Arch Linux

```
>yay -S topologic-git
-To use in program use
>#include <topologic/topologic.h>
-Link with
>gcc ... -ltopologic
```

#### 1.1.2 Manualy

```
Build C Library >make
-creates libtopologic.a
Build C++ Library >make cpp
-creates libtopologic.a
Build Python 3 Library >make python
-creates topylogic/topylogic.so
Build Python 2 Library >make python2
-creates topylogic/topylogic.so
Clean >make clean
Debug >make CFLAGS=-DDEBUG
```

## 1.2 TODO

-C# Wrap

-Rust Wrap

-Python (2/3) Wrap

## 1.3 Errata

### 1.3.1 Edge/vertex modification

Modifying/deleting vertices and edges inside `f()` not using submit request can lead to undefined behavior or dead locks. This is because in `CONTEXT` set to `SWITCH` or `NONE` with many starting vertices, can lead to a structure being `NULL'd` with its lock destroyed while another thread is holding that lock or two threads trying to modify each other and thus resulting in a dead lock. It is up to the client's discretion to modify directly when in `SWITCH` or `NONE` with many starting vertices. Although in `SINGLE`, this should be fine. To mitigate this problem, the client should use submit request which will handle the requests sequentially with destroying structures last.

### 1.3.2 Graph modification

Trying to delete the graph while running will result in undefined behavior. Destroying the graph does not lock any thread and thus will cause race conditions. The graph should be deleted only once all threads reach a sink.

### 1.3.3 Parameter Passing

Parameters passed to edges or vertices functions will be free'd immediately after use and therefore will cause an error should they try and be accessed. To mitigate the values may be stored in the edge's or vertex's global or shared variables. Any non standard data type, such as struct, is dependent on the user to free its content as the library will only free the pointer to the struct and the pointer to the array of variables.

### 1.3.4 Illegal Arguments

Passing wrong values or wrong number of variables to any function will result in failure. A client should be aware of which vertices connect and what edges it has and the proper handling required between such connections. Should the client choose to dynamically add/remove vertices or edges or even modify while running should be aware of the changes that may occur in the graph and the resulting change in dependence on proper variable handling.

### 1.3.5 Graph Modification (cont.)

It is possible to modify the graph while it is running. To do so the client should submit request to add a change or pause the graph. Making any changes externally may result in undefined behavior if done improperly. Modifying the graph directly while running could result in failure. Should the number of init vertex args not much the number of vertices in the start set the program will result in an error.

### 1.3.6 Data Structure Errata

Stacks and [AVLTree](#) can take non malloc'd data and function normally within scope of those non malloc'd data. However, in another scope the memory will be unaddressable and thus should be malloc'd memory instead. The stack and AVL Tree will not free the void \* data since the void \* data structure is unknown to them and thus the client should free the memory.

### 1.3.7 SWITCH\_UNSAFE

Should SWITCH\_USNAFE be used as the context, then the graph will run like how it runs in SWITCH, but the variables shared between the edge and the second vertex b will be passed to the edges function. Trying to read the shared variables may lead to a race condition in which vertex b or one/some of it's edges are modifying those varibales. The client is expected to handle such race conditions, and failing to do so can lead to a possibly unsafe execution. Thus this mode is UNSAFE. It is safe to use the shared variables with vertex b in context modes NONE and SINGLE since in NONE vertex b cannot also be active, and in SINGLE there is no threading.



## Chapter 2

# Data Structure Index

### 2.1 Data Structures

Here are the data structures with brief descriptions:

AVLNode	9
AVLTree	10
destroy_edge_id_request	11
destroy_edge_request	12
destroy_vertex_id_request	12
destroy_vertex_request	13
edge	14
edge_request	16
fireable	17
graph	18
mod_edge_vars_request	23
mod_vertex_request	24
request	25
shared_edge	26
stack	26
stack_node	27
vertex	28
vertex_request	30
vertex_result	31





## Chapter 3

# File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

include/AVL.h . . . . .	33
include/context.h . . . . .	35
include/edge.h . . . . .	37
include/graph.h . . . . .	37
include/header.h . . . . .	40
include/request.h . . . . .	40
include/stack.h . . . . .	41
include/topologic.h . . . . .	42
include/vertex.h . . . . .	55
src/AVL.c . . . . .	56
src/edge.c . . . . .	62
src/graph.c . . . . .	64
src/request.c . . . . .	66
src/stack.c . . . . .	68
src/topologic.c . . . . .	69
src/topologic_json.c . . . . .	72
src/vertex.c . . . . .	73



## Chapter 4

# Data Structure Documentation

### 4.1 AVLNode Struct Reference

```
#include <AVL.h>
```

#### Data Fields

- void \* [data](#)
- int [id](#)
- int [height](#)
- struct [AVLNode](#) \* [left](#)
- struct [AVLNode](#) \* [right](#)

#### 4.1.1 Field Documentation

##### 4.1.1.1 data

```
void* AVLNode::data
```

Referenced by `create_node()`, `destroy_avl_nodes()`, `find_node()`, `inorder_nodes()`, `insert()`, `postorder_nodes()`, `preorder_nodes()`, `remove_ID()`, `remove_node()`, and `stackify_nodes()`.

##### 4.1.1.2 height

```
int AVLNode::height
```

Referenced by `balance()`, `create_node()`, `destroy_avl_nodes()`, `insert_node()`, `left_rotate()`, `max_height()`, `remove_node()`, `right_rotate()`, and `stackify_nodes()`.

#### 4.1.1.3 id

```
int AVLNode::id
```

Referenced by `create_node()`, `find_node()`, `inorder_nodes()`, `insert_node()`, `postorder_nodes()`, `preorder_nodes()`, and `remove_node()`.

#### 4.1.1.4 left

```
struct AVLNode* AVLNode::left
```

Referenced by `balance()`, `create_node()`, `destroy_avl_nodes()`, `find_node()`, `inorder_nodes()`, `insert_node()`, `left_rotate()`, `max_height()`, `minNode()`, `postorder_nodes()`, `preorder_nodes()`, `remove_node()`, `right_rotate()`, and `stackify_nodes()`.

#### 4.1.1.5 right

```
struct AVLNode* AVLNode::right
```

Referenced by `balance()`, `create_node()`, `destroy_avl_nodes()`, `find_node()`, `inorder_nodes()`, `insert_node()`, `left_rotate()`, `max_height()`, `postorder_nodes()`, `preorder_nodes()`, `remove_node()`, `right_rotate()`, and `stackify_nodes()`.

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

- `include/AVL.h`

## 4.2 AVLTree Struct Reference

```
#include <AVL.h>
```

### Data Fields

- struct `AVLNode` \* `root`
- int `size`

#### 4.2.1 Field Documentation

#### 4.2.1.1 root

```
struct AVLNode* AVLTree::root
```

Referenced by `destroy_avl()`, `find()`, `init_avl()`, `inorder()`, `insert()`, `postorder()`, `preorder()`, `remove_ID()`, and `stackify()`.

#### 4.2.1.2 size

```
int AVLTree::size
```

Referenced by `destroy_avl()`, `init_avl()`, `inorder()`, `insert()`, `postorder()`, `preorder()`, `print_graph()`, `print_state()`, `remove_ID()`, and `stackify()`.

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

- `include/AVL.h`

## 4.3 destroy\_edge\_id\_request Struct Reference

```
#include <edge.h>
```

### Data Fields

- struct `vertex` \* `a`
- int `id`

#### 4.3.1 Field Documentation

##### 4.3.1.1 a

```
struct vertex* destroy_edge_id_request::a
```

Referenced by `procces_request()`.

##### 4.3.1.2 id

```
int destroy_edge_id_request::id
```

Referenced by `procces_request()`.

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

- `include/edge.h`

## 4.4 `destroy_edge_request` Struct Reference

```
#include <edge.h>
```

### Data Fields

- struct `vertex` \* `a`
- struct `vertex` \* `b`

#### 4.4.1 Field Documentation

##### 4.4.1.1 `a`

```
struct vertex* destroy_edge_request::a
```

Referenced by `procces_request()`.

##### 4.4.1.2 `b`

```
struct vertex* destroy_edge_request::b
```

Referenced by `procces_request()`.

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

- `include/edge.h`

## 4.5 `destroy_vertex_id_request` Struct Reference

```
#include <vertex.h>
```

### Data Fields

- struct `graph` \* `graph`
- int `id`

#### 4.5.1 Field Documentation

#### 4.5.1.1 graph

```
struct graph* destroy_vertex_id_request::graph
```

Referenced by `procces_request()`.

#### 4.5.1.2 id

```
int destroy_vertex_id_request::id
```

Referenced by `procces_request()`.

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

- `include/vertex.h`

## 4.6 destroy\_vertex\_request Struct Reference

```
#include <vertex.h>
```

### Data Fields

- struct [graph](#) \* [graph](#)
- struct [vertex](#) \* [vertex](#)

### 4.6.1 Field Documentation

#### 4.6.1.1 graph

```
struct graph* destroy_vertex_request::graph
```

Referenced by `procces_request()`.

#### 4.6.1.2 vertex

```
struct vertex* destroy_vertex_request::vertex
```

Referenced by `procces_request()`.

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

- `include/vertex.h`

## 4.7 edge Struct Reference

```
#include <edge.h>
```

### Data Fields

- int [id](#)
- int(\* [f](#))(void \*, void \*, const void \*const [a\\_vars](#), const void \*const [b\\_vars](#))
- void \* [glbl](#)
- const void \*const \* [a\\_vars](#)
- const void \*const \* [b\\_vars](#)
- struct [vertex](#) \* [a](#)
- struct [vertex](#) \* [b](#)
- enum [edge\\_type](#) [edge\\_type](#)
- struct [edge](#) \* [bi\\_edge](#)
- pthread\_mutex\_t [bi\\_edge\\_lock](#)

### 4.7.1 Detailed Description

Edge

### 4.7.2 Field Documentation

#### 4.7.2.1 [a](#)

```
struct vertex* edge::a
```

Referenced by [create\\_bi\\_edge\(\)](#), [create\\_edge\(\)](#), [modify\\_bi\\_edge\(\)](#), [modify\\_edge\(\)](#), [print\\_edges\(\)](#), [remove\\_bi\\_↔edge\(\)](#), [remove\\_edge\(\)](#), [remove\\_edge\\_id\(\)](#), and [remove\\_vertex\(\)](#).

#### 4.7.2.2 [a\\_vars](#)

```
const void* const* edge::a_vars
```

Referenced by [create\\_edge\(\)](#), [fire\(\)](#), [remove\\_edge\(\)](#), [remove\\_edge\\_id\(\)](#), and [run\\_single\(\)](#).

#### 4.7.2.3 [b](#)

```
struct vertex* edge::b
```

Referenced by [create\\_bi\\_edge\(\)](#), [create\\_edge\(\)](#), [fire\(\)](#), [modify\\_bi\\_edge\(\)](#), [modify\\_edge\(\)](#), [print\\_edges\(\)](#), [remove\\_↔\\_bi\\_edge\(\)](#), [remove\\_edge\(\)](#), [remove\\_edge\\_id\(\)](#), [remove\\_vertex\(\)](#), and [run\\_single\(\)](#).



#### 4.7.2.4 b\_vars

```
const void* const* edge::b_vars
```

Referenced by `create_edge()`, `fire()`, `remove_edge()`, `remove_edge_id()`, and `run_single()`.

#### 4.7.2.5 bi\_edge

```
struct edge* edge::bi_edge
```

Referenced by `create_bi_edge()`, `create_edge()`, `print_edges()`, `remove_edge()`, `remove_edge_id()`, and `remove_vertex()`.

#### 4.7.2.6 bi\_edge\_lock

```
pthread_mutex_t edge::bi_edge_lock
```

Referenced by `create_bi_edge()`, `fire()`, `remove_edge()`, `remove_edge_id()`, and `remove_vertex()`.

#### 4.7.2.7 edge\_type

```
enum edge_type edge::edge_type
```

Referenced by `create_bi_edge()`, `create_edge()`, `fire()`, `print_edges()`, `remove_edge()`, `remove_edge_id()`, and `remove_vertex()`.

#### 4.7.2.8 f

```
int(* edge::f) (void *, void *, const void *const a_vars, const void *const b_vars)
```

Referenced by `create_bi_edge()`, `create_edge()`, `fire()`, `modify_bi_edge()`, `modify_edge()`, `print_edges()`, `remove_edge()`, `remove_edge_id()`, `remove_vertex()`, and `run_single()`.

#### 4.7.2.9 glbl

```
void* edge::glbl
```

Referenced by `create_bi_edge()`, `create_edge()`, `fire()`, `modify_bi_edge()`, `modify_edge()`, `print_edges()`, `print_state()`, `remove_edge()`, `remove_edge_id()`, `remove_vertex()`, and `run_single()`.

#### 4.7.2.10 id

```
int edge::id
```

Referenced by `create_edge()`, `print_edges()`, `remove_edge()`, `remove_edge_id()`, and `remove_vertex()`.

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

- `include/edge.h`

## 4.8 edge\_request Struct Reference

```
#include <edge.h>
```

### Data Fields

- struct `vertex` \* `a`
- struct `vertex` \* `b`
- `int(* f)(void *, void *, const void *const, const void *const)`
- `void * glbl`

### 4.8.1 Field Documentation

#### 4.8.1.1 a

```
struct vertex* edge_request::a
```

Referenced by `procces_request()`.

#### 4.8.1.2 b

```
struct vertex* edge_request::b
```

Referenced by `procces_request()`.

#### 4.8.1.3 f

```
int(* edge_request::f)(void *, void *, const void *const, const void *const)
```

Referenced by `procces_request()`.

#### 4.8.1.4 glbl

```
void* edge_request::glbl
```

Referenced by `procces_request()`.

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

- `include/edge.h`

## 4.9 fireable Struct Reference

```
#include <topologic.h>
```

### Data Fields

- struct `graph` \* `graph`
- struct `vertex` \* `vertex`
- struct `vertex_result` \* `args`
- enum `STATES` `color`
- int `iloop`

### 4.9.1 Field Documentation

#### 4.9.1.1 args

```
struct vertex_result* fireable::args
```

Referenced by `fire()`, `fire_pthread()`, `run()`, and `switch_vertex()`.

#### 4.9.1.2 color

```
enum STATES fireable::color
```

Referenced by `fire()`, `fire_pthread()`, `run()`, and `switch_vertex()`.

#### 4.9.1.3 graph

```
struct graph* fireable::graph
```

Referenced by `fire_pthread()`, `run()`, and `switch_vertex()`.

#### 4.9.1.4 iloop

```
int fireable::iloop
```

Referenced by `fire()`, `fire_pthread()`, `run()`, and `switch_vertex()`.

#### 4.9.1.5 vertex

```
struct vertex* fireable::vertex
```

Referenced by `fire_pthread()`, `run()`, and `switch_vertex()`.

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

- `include/topologic.h`

## 4.10 graph Struct Reference

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

### Data Fields

- enum `CONTEXT` `context`
- enum `MEM_OPTION` `mem_option`
- struct `AVLTree` \* `vertices`
- struct `stack` \* `start`
- struct `stack` \* `modify`
- struct `stack` \* `remove_edges`
- struct `stack` \* `remove_vertices`
- int `max_state_changes`
- int `max_loop`
- int `snapshot_timestamp`
- unsigned int `lvl_verbose`
- int `state_count`
- pthread\_mutex\_t `lock`
- pthread\_mutex\_t `color_lock`
- sig\_atomic\_t `state`
- sig\_atomic\_t `previous_color`
- sig\_atomic\_t `print_flag`
- sig\_atomic\_t `red_vertex_count`
- sig\_atomic\_t `black_vertex_count`
- sig\_atomic\_t `pause`
- sig\_atomic\_t `red_locked`
- sig\_atomic\_t `black_locked`
- sig\_atomic\_t `num_vertices`
- pthread\_cond\_t `pause_cond`
- pthread\_cond\_t `red_fire`
- pthread\_cond\_t `black_fire`

### 4.10.1 Detailed Description

Graph

### 4.10.2 Field Documentation

#### 4.10.2.1 black\_fire

```
pthread_cond_t graph::black_fire
```

Referenced by `destroy_graph()`, `fire()`, `graph_init()`, and `run()`.

#### 4.10.2.2 black\_locked

```
sig_atomic_t graph::black_locked
```

Referenced by `destroy_graph()`, `fire()`, `graph_init()`, and `run()`.

#### 4.10.2.3 black\_vertex\_count

```
sig_atomic_t graph::black_vertex_count
```

Referenced by `destroy_graph()`, `fire()`, `graph_init()`, and `run()`.

#### 4.10.2.4 color\_lock

```
pthread_mutex_t graph::color_lock
```

Referenced by `destroy_graph()`, `graph_init()`, and `run()`.

#### 4.10.2.5 context

```
enum CONTEXT graph::context
```

Referenced by `create_vertex()`, `destroy_graph()`, `fire()`, `graph_init()`, `print_graph()`, `process_requests()`, `remove_vertex()`, `remove_vertex_id()`, `run()`, `run_single()`, `start_set()`, and `submit_request()`.

#### 4.10.2.6 lock

```
pthread_mutex_t graph::lock
```

Referenced by `create_vertex()`, `destroy_graph()`, `fire()`, `graph_init()`, `pause_graph()`, `print_graph()`, `process_requests()`, `remove_vertex()`, `remove_vertex_id()`, `resume_graph()`, `run()`, `run_single()`, and `submit_request()`.

#### 4.10.2.7 lvl\_verbose

```
unsigned int graph::lvl_verbose
```

Referenced by `graph_init()`, `print_edges()`, `print_graph()`, and `print_state()`.

#### 4.10.2.8 max\_loop

```
int graph::max_loop
```

Referenced by `fire()`, `graph_init()`, `print_graph()`, and `run_single()`.

#### 4.10.2.9 max\_state\_changes

```
int graph::max_state_changes
```

Referenced by `fire()`, `graph_init()`, `print_graph()`, `run()`, and `run_single()`.

#### 4.10.2.10 mem\_option

```
enum MEM_OPTION graph::mem_option
```

Referenced by `graph_init()`, `run()`, and `switch_vertex()`.

#### 4.10.2.11 modify

```
struct stack* graph::modify
```

Referenced by `destroy_graph()`, `graph_init()`, `process_requests()`, and `submit_request()`.

#### 4.10.2.12 num\_vertices

```
sig_atomic_t graph::num_vertices
```

Referenced by `fire()`, `graph_init()`, `print_graph()`, `run()`, and `run_single()`.

#### 4.10.2.13 pause

```
sig_atomic_t graph::pause
```

Referenced by `graph_init()`, `pause_graph()`, `resume_graph()`, `run()`, and `run_single()`.

#### 4.10.2.14 pause\_cond

```
pthread_cond_t graph::pause_cond
```

Referenced by `destroy_graph()`, `graph_init()`, `pause_graph()`, `run()`, and `run_single()`.

#### 4.10.2.15 previous\_color

```
sig_atomic_t graph::previous_color
```

Referenced by `graph_init()`, and `run()`.

#### 4.10.2.16 print\_flag

```
sig_atomic_t graph::print_flag
```

Referenced by `graph_init()`, and `run()`.

#### 4.10.2.17 red\_fire

```
pthread_cond_t graph::red_fire
```

Referenced by `destroy_graph()`, `fire()`, `graph_init()`, and `run()`.

#### 4.10.2.18 red\_locked

```
sig_atomic_t graph::red_locked
```

Referenced by `destroy_graph()`, `fire()`, `graph_init()`, and `run()`.

#### 4.10.2.19 red\_vertex\_count

```
sig_atomic_t graph::red_vertex_count
```

Referenced by `destroy_graph()`, `fire()`, `graph_init()`, and `run()`.

#### 4.10.2.20 remove\_edges

```
struct stack* graph::remove_edges
```

Referenced by `destroy_graph()`, `graph_init()`, `process_requests()`, and `submit_request()`.

#### 4.10.2.21 remove\_vertices

```
struct stack* graph::remove_vertices
```

Referenced by `destroy_graph()`, `graph_init()`, `process_requests()`, and `submit_request()`.

#### 4.10.2.22 snapshot\_timestamp

```
int graph::snapshot_timestamp
```

Referenced by `graph_init()`, and `print_graph()`.

#### 4.10.2.23 start

```
struct stack* graph::start
```

Referenced by `destroy_graph()`, `graph_init()`, `run()`, `run_single()`, and `start_set()`.



#### 4.10.2.24 state

```
sig_atomic_t graph::state
```

Referenced by `destroy_graph()`, `fire()`, `graph_init()`, `run()`, and `run_single()`.

#### 4.10.2.25 state\_count

```
int graph::state_count
```

Referenced by `fire()`, `graph_init()`, `print_graph()`, `run()`, and `run_single()`.

#### 4.10.2.26 vertices

```
struct AVLTree* graph::vertices
```

Referenced by `create_vertex()`, `destroy_graph()`, `graph_init()`, `print_graph()`, `print_state()`, `remove_vertex()`, `remove_vertex_id()`, and `start_set()`.

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

- `include/graph.h`

## 4.11 mod\_edge\_vars\_request Struct Reference

```
#include <vertex.h>
```

### Data Fields

- struct `vertex` \* `vertex`
- void \* `edge_vars`

#### 4.11.1 Field Documentation

##### 4.11.1.1 edge\_vars

```
void* mod_edge_vars_request::edge_vars
```

Referenced by `procces_request()`.

#### 4.11.1.2 vertex

```
struct vertex* mod_edge_vars_request::vertex
```

Referenced by `procces_request()`.

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

- `include/vertex.h`

## 4.12 mod\_vertex\_request Struct Reference

```
#include <vertex.h>
```

### Data Fields

- struct [vertex](#) \* [vertex](#)
- void(\* [f](#))(struct [graph](#) \*, struct [vertex\\_result](#) \*, void \*, void \*)
- void \* [glbl](#)

### 4.12.1 Field Documentation

#### 4.12.1.1 f

```
void(* mod_vertex_request::f) (struct graph *, struct vertex\_result *, void *, void *)
```

Referenced by `procces_request()`.

#### 4.12.1.2 glbl

```
void* mod_vertex_request::glbl
```

Referenced by `procces_request()`.

#### 4.12.1.3 vertex

```
struct vertex* mod_vertex_request::vertex
```

Referenced by `procces_request()`.

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

- `include/vertex.h`

## 4.13 request Struct Reference

```
#include <request.h>
```

### Data Fields

- enum [REQUESTS request](#)
- void(\* [f](#))(void \*)
- void \* [args](#)

### 4.13.1 Detailed Description

Request

### 4.13.2 Field Documentation

#### 4.13.2.1 args

```
void* request::args
```

Referenced by `create_request()`, `destroy_request()`, and `procces_request()`.

#### 4.13.2.2 f

```
void(* request::f) (void *)
```

Referenced by `create_request()`, `destroy_request()`, and `procces_request()`.

#### 4.13.2.3 request

```
enum REQUESTS request::request
```

Referenced by `create_request()`, `destroy_request()`, `procces_request()`, and `submit_request()`.

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

- `include/request.h`

## 4.14 shared\_edge Union Reference

```
#include <vertex.h>
```

### Data Fields

- void \* [vertex\\_data](#)
- const void \*const \* [edge\\_data](#)

### 4.14.1 Field Documentation

#### 4.14.1.1 edge\_data

```
const void* const* shared_edge::edge_data
```

Referenced by `create_edge()`.

#### 4.14.1.2 vertex\_data

```
void* shared_edge::vertex_data
```

Referenced by `create_vertex()`, `fire()`, `modify_shared_edge_vars()`, `remove_vertex()`, and `run_single()`.

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

- `include/vertex.h`

## 4.15 stack Struct Reference

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

### Data Fields

- struct [stack\\_node](#) \* [root](#)
- int [length](#)

### 4.15.1 Field Documentation

#### 4.15.1.1 `length`

```
int stack::length
```

Referenced by `destroy_stack()`, `get()`, `init_stack()`, `pop()`, `print_edges()`, `print_state()`, `push()`, and `run_single()`.

#### 4.15.1.2 `root`

```
struct stack_node* stack::root
```

Referenced by `destroy_stack()`, `get()`, `init_stack()`, `pop()`, and `push()`.

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

- `include/stack.h`

## 4.16 `stack_node` Struct Reference

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

### Data Fields

- void \* `data`
- struct `stack_node` \* `next`

#### 4.16.1 Field Documentation

##### 4.16.1.1 `data`

```
void* stack_node::data
```

Referenced by `destroy_stack()`, `get()`, `pop()`, and `push()`.

##### 4.16.1.2 `next`

```
struct stack_node* stack_node::next
```

Referenced by `destroy_stack()`, `get()`, `pop()`, and `push()`.

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

- `include/stack.h`

## 4.17 vertex Struct Reference

```
#include <vertex.h>
```

### Data Fields

- int `id`
- int `is_active`
- void(\* `f`)(struct `graph` \*, struct `vertex_result` \*, void \*, void \*)
- void \* `glbl`
- union `shared_edge` \* `shared`
- pthread\_mutex\_t `lock`
- struct `AVLTree` \* `edge_tree`
- struct `AVLTree` \* `joining_vertices`
- enum `CONTEXT` `context`

### 4.17.1 Detailed Description

vertex

### 4.17.2 Field Documentation

#### 4.17.2.1 context

```
enum CONTEXT vertex::context
```

Referenced by `create_bi_edge()`, `create_edge()`, `create_vertex()`, `modify_edge()`, `modify_shared_edge_vars()`, `modify_vertex()`, `remove_edge()`, `remove_edge_id()`, and `remove_vertex()`.

#### 4.17.2.2 edge\_tree

```
struct AVLTree* vertex::edge_tree
```

Referenced by `create_edge()`, `create_vertex()`, `fire()`, `modify_edge()`, `print_state()`, `remove_edge()`, `remove_edge_id()`, `remove_vertex()`, and `run_single()`.

#### 4.17.2.3 f

```
void(* vertex::f) (struct graph *, struct vertex_result *, void *, void *)
```

Referenced by `create_vertex()`, `fire()`, `modify_vertex()`, `print_state()`, and `run_single()`.

#### 4.17.2.4 glbl

```
void* vertex::glbl
```

Referenced by `create_vertex()`, `fire()`, `modify_vertex()`, `print_state()`, `remove_vertex()`, and `run_single()`.

#### 4.17.2.5 id

```
int vertex::id
```

Referenced by `create_edge()`, `create_vertex()`, `modify_edge()`, `print_state()`, `remove_edge()`, `remove_edge_id()`, `remove_vertex()`, and `start_set()`.

#### 4.17.2.6 is\_active

```
int vertex::is_active
```

Referenced by `create_vertex()`, `fire()`, `print_state()`, `run()`, and `run_single()`.

#### 4.17.2.7 joining\_vertices

```
struct AVLTree* vertex::joining_vertices
```

Referenced by `create_edge()`, `create_vertex()`, `remove_edge()`, `remove_edge_id()`, and `remove_vertex()`.

#### 4.17.2.8 lock

```
pthread_mutex_t vertex::lock
```

Referenced by `create_edge()`, `create_vertex()`, `fire()`, `modify_edge()`, `modify_shared_edge_vars()`, `modify_vertex()`, `remove_edge()`, `remove_edge_id()`, and `remove_vertex()`.

#### 4.17.2.9 shared

```
union shared_edge* vertex::shared
```

Referenced by `create_edge()`, `create_vertex()`, `fire()`, `modify_shared_edge_vars()`, `print_state()`, `remove_vertex()`, and `run_single()`.

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

- `include/vertex.h`

## 4.18 vertex\_request Struct Reference

```
#include <vertex.h>
```

### Data Fields

- struct [graph](#) \* [graph](#)
- int [id](#)
- void(\* [f](#))(struct [graph](#) \*, struct [vertex\\_result](#) \*, void \*, void \*)
- void \* [glbl](#)

### 4.18.1 Field Documentation

#### 4.18.1.1 f

```
void(* vertex_request::f) (struct graph *, struct vertex\_result *, void *, void *)
```

Referenced by `procces_request()`.

#### 4.18.1.2 glbl

```
void* vertex_request::glbl
```

Referenced by `procces_request()`.

#### 4.18.1.3 graph

```
struct graph* vertex_request::graph
```

Referenced by `procces_request()`.

#### 4.18.1.4 id

```
int vertex_request::id
```

Referenced by `procces_request()`.

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

- `include/vertex.h`



## 4.19 vertex\_result Struct Reference

```
#include <vertex.h>
```

### Data Fields

- void \* [vertex\\_argv](#)
- size\_t [vertex\\_size](#)
- void \* [edge\\_argv](#)
- size\_t [edge\\_size](#)

### 4.19.1 Detailed Description

[vertex\\_result](#)

### 4.19.2 Field Documentation

#### 4.19.2.1 edge\_argv

```
void* vertex_result::edge_argv
```

Referenced by `fire()`, `run_single()`, and `switch_vertex()`.

#### 4.19.2.2 edge\_size

```
size_t vertex_result::edge_size
```

Referenced by `switch_vertex()`.

#### 4.19.2.3 vertex\_argv

```
void* vertex_result::vertex_argv
```

Referenced by `fire()`, `run_single()`, and `switch_vertex()`.

#### 4.19.2.4 vertex\_size

```
size_t vertex_result::vertex_size
```

Referenced by `switch_vertex()`.

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

- `include/vertex.h`



## Chapter 5

# File Documentation

### 5.1 include/AVL.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include "../stack.h"
```

#### Data Structures

- struct [AVLNode](#)
- struct [AVLTree](#)

#### Functions

- struct [AVLTree](#) \* [init\\_avl](#) ()
- int [insert](#) (struct [AVLTree](#) \*tree, void \*data, int id)
- void \* [remove\\_ID](#) (struct [AVLTree](#) \*tree, int id)
- void \* [find](#) (struct [AVLTree](#) \*tree, int id)
- void [inorder](#) (struct [AVLTree](#) \*tree, struct [stack](#) \*stack)
- void [postorder](#) (struct [AVLTree](#) \*tree, struct [stack](#) \*stack)
- void [preorder](#) (struct [AVLTree](#) \*tree, struct [stack](#) \*stack)
- void [stackify](#) (struct [AVLTree](#) \*tree, struct [stack](#) \*stack)
- void [destroy\\_avl](#) (struct [AVLTree](#) \*tree)

#### 5.1.1 Function Documentation

##### 5.1.1.1 [destroy\\_avl\(\)](#)

```
void destroy_avl (
    struct AVLTree * tree )
```

References [destroy\\_avl\\_nodes\(\)](#), [AVLTree::root](#), and [AVLTree::size](#).

Referenced by [create\\_vertex\(\)](#), [destroy\\_graph\\_avl\(\)](#), and [graph\\_init\(\)](#).

#### 5.1.1.2 find()

```
void* find (
    struct AVLTree * tree,
    int id )
```

References find\_node(), and AVLTree::root.

Referenced by create\_edge(), modify\_edge(), remove\_vertex\_id(), and start\_set().

#### 5.1.1.3 init\_avl()

```
struct AVLTree* init_avl ( )
```

References AVLTree::root, and AVLTree::size.

Referenced by create\_vertex(), and graph\_init().

#### 5.1.1.4 inorder()

```
void inorder (
    struct AVLTree * tree,
    struct stack * stack )
```

References inorder\_nodes(), AVLTree::root, and AVLTree::size.

Referenced by print\_state().

#### 5.1.1.5 insert()

```
int insert (
    struct AVLTree * tree,
    void * data,
    int id )
```

References create\_node(), AVLNode::data, insert\_node(), AVLTree::root, and AVLTree::size.

Referenced by create\_edge(), create\_vertex(), and insert\_node().

### 5.1.1.6 postorder()

```
void postorder (
    struct AVLTree * tree,
    struct stack * stack )
```

References postorder\_nodes(), AVLTree::root, and AVLTree::size.

### 5.1.1.7 preorder()

```
void preorder (
    struct AVLTree * tree,
    struct stack * stack )
```

References preorder\_nodes(), AVLTree::root, and AVLTree::size.

Referenced by destroy\_graph\_avl(), fire(), print\_edges(), and run\_single().

### 5.1.1.8 remove\_ID()

```
void* remove_ID (
    struct AVLTree * tree,
    int id )
```

References AVLNode::data, remove\_node(), AVLTree::root, and AVLTree::size.

Referenced by create\_edge(), remove\_edge(), remove\_edge\_id(), and remove\_vertex().

### 5.1.1.9 stackify()

```
void stackify (
    struct AVLTree * tree,
    struct stack * stack )
```

References AVLTree::root, AVLTree::size, and stackify\_nodes().

Referenced by remove\_vertex().

## 5.2 include/context.h File Reference

### Enumerations

- enum `CONTEXT` { `NONE` = 0 , `SINGLE` = 1 , `SWITCH` = 2 , `SWITCH_UNSAFE` = 3 }

## 5.2.1 Enumeration Type Documentation

### 5.2.1.1 CONTEXT

enum `CONTEXT`

Enum for how the graph handles context switches, or not at all NONE: First valid edge is taken only and the process does not change SINGLE: Is the same as NONE but only one vertex may be selected as start SWITCH: All valid edges are taken. A process per vertex is spawned and previous process is killed. SWITCH\_UNSAFE: Same as SWITCH but will pass edge->b\_vars to edge->f

## Enumerator

NONE	
SINGLE	
SWITCH	
SWITCH_UNSAFE	

## 5.3 include/edge.h File Reference

```
#include <stdio.h>
#include "../vertex.h"
```

### Data Structures

- struct [edge](#)
- struct [edge\\_request](#)
- struct [destroy\\_edge\\_request](#)
- struct [destroy\\_edge\\_id\\_request](#)

### Enumerations

- enum [edge\\_type](#) { [EDGE](#) = 0 , [BI\\_EDGE](#) = 1 , [SELF\\_EDGE](#) = 2 }

#### 5.3.1 Enumeration Type Documentation

##### 5.3.1.1 edge\_type

```
enum edge\_type
```

## Enumerator

EDGE	
BI_EDGE	
SELF_EDGE	

## 5.4 include/graph.h File Reference

```
#include <pthread.h>
#include <signal.h>
```

```
#include "../context.h"
```

## Data Structures

- struct `graph`

## Macros

- #define `MAX_ATTEMPTS` 4
- #define `THREAD_ATTEMPT_SLEEP` 3

## Enumerations

- enum `MEM_OPTION` { `ABORT` = 0 , `WAIT` = 1 , `CONTINUE` = 2 }
- enum `STATES` { `PRINT` = 0 , `RED` = 1 , `BLACK` = 2 , `TERMINATE` = 3 }
- enum `SNAPSHOT` { `NO_SNAP` = -1 , `START_STOP` = 0 }
- enum `VERBOSITY` {  
    `NO_VERB` = 0 , `VERTICES` = 1 , `EDGES` = 2 , `FUNCTIONS` = 4 ,  
    `GLOBALS` = 8 }

### 5.4.1 Macro Definition Documentation

#### 5.4.1.1 MAX\_ATTEMPTS

```
#define MAX_ATTEMPTS 4
```

#### 5.4.1.2 THREAD\_ATTEMPT\_SLEEP

```
#define THREAD_ATTEMPT_SLEEP 3
```

### 5.4.2 Enumeration Type Documentation

#### 5.4.2.1 MEM\_OPTION

```
enum MEM_OPTION
```

Enum for memory handling when forking – if mem amount is exceeded ABORT: Kill the program and clean WAIT: Wait until there is enough memory and try again CONTINUE: Ignore edge error and move on



**Enumerator**

ABORT	
WAIT	
CONTINUE	

**5.4.2.2 SNAPSHOT**

enum [SNAPSHOT](#)

Enum for snapshots. NONE: Record nothing START\_STOP: Record first and last state

**Enumerator**

NO_SNAP	
START_STOP	

**5.4.2.3 STATES**

enum [STATES](#)

Enum for state of global manager – locking when printing and firing; Ensures proper locking between printing node information and firing information PRINT: Print, then lock RED: # Readers = 0 -> unlock print, goes to BLACK at finish BLACK: Wait until # Readers -> 0, then print

**Enumerator**

PRINT	
RED	
BLACK	
TERMINATE	

**5.4.2.4 VERBOSITY**

enum [VERBOSITY](#)

Enum for how verbose the records are NONE: Record nothing NODES: Record nodes EDGES: Record edges FUNCTIONS: Record the functions of nodes and/or edges GLOBALS: Record the globals of nodes and/or edges; Also will record shared edges

## Enumerator

NO_VERB	
VERTICES	
EDGES	
FUNCTIONS	
GLOBALS	

## 5.5 include/header.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
#include <signal.h>
#include <pthread.h>
#include <unistd.h>
#include <time.h>
```

## 5.6 include/request.h File Reference

### Data Structures

- struct [request](#)

### Enumerations

- enum [REQUESTS](#) {  
[CREAT\\_VERTEX](#) = 0 , [CREAT\\_EDGE](#) = 1 , [CREAT\\_BI\\_EDGE](#) = 2 , [MOD\\_VERTEX](#) = 3 ,  
[MOD\\_EDGE\\_VARS](#) = 4 , [MOD\\_EDGE](#) = 5 , [MOD\\_BI\\_EDGE](#) = 6 , [DESTROY\\_VERTEX](#) = 7 ,  
[DESTROY\\_VERTEX\\_BY\\_ID](#) = 8 , [DESTROY\\_EDGE](#) = 9 , [DESTROY\\_BI\\_EDGE](#) = 10 , [DESTROY\\_EDGE\\_BY\\_ID](#)  
= 11 ,  
[GENERIC](#) = 12 }

### 5.6.1 Enumeration Type Documentation

#### 5.6.1.1 REQUESTS

enum [REQUESTS](#)

Enum for submitting a request to be handles MODIFY: Modify values in existing edges or vertices as well as add vertices or edges DESTROY\_VERTEX: Remove vertex from graph DESTROY\_EDGE: Remove edge from graph

## Enumerator

CREAT_VERTEX	
CREAT_EDGE	
CREAT_BI_EDGE	
MOD_VERTEX	
MOD_EDGE_VARS	
MOD_EDGE	
MOD_BI_EDGE	
DESTROY_VERTEX	
DESTROY_VERTEX_BY_ID	
DESTROY_EDGE	
DESTROY_BI_EDGE	
DESTROY_EDGE_BY_ID	
GENERIC	

## 5.7 include/stack.h File Reference

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

### Data Structures

- struct [stack\\_node](#)
- struct [stack](#)

### Functions

- struct [stack](#) \* [init\\_stack](#) ()
- void \* [get](#) (struct [stack](#) \*[stack](#), int index)
- void \* [pop](#) (struct [stack](#) \*[stack](#))
- int [push](#) (struct [stack](#) \*[stack](#), void \*data)
- void [destroy\\_stack](#) (struct [stack](#) \*[stack](#))

### 5.7.1 Function Documentation

#### 5.7.1.1 [destroy\\_stack\(\)](#)

```
void destroy_stack (
    struct stack * stack )
```

References [stack\\_node::data](#), [stack::length](#), [stack\\_node::next](#), and [stack::root](#).

Referenced by [destroy\\_graph\\_avl\(\)](#), [destroy\\_graph\\_stack\(\)](#), [fire\(\)](#), [graph\\_init\(\)](#), [print\\_edges\(\)](#), [print\\_state\(\)](#), [remove\\_vertex\(\)](#), and [run\\_single\(\)](#).

### 5.7.1.2 get()

```
void* get (
    struct stack * stack,
    int index )
```

References `stack_node::data`, `stack::length`, `stack_node::next`, and `stack::root`.

### 5.7.1.3 init\_stack()

```
struct stack* init_stack ( )
```

References `stack::length`, and `stack::root`.

Referenced by `destroy_graph_avl()`, `fire()`, `graph_init()`, `print_edges()`, `print_state()`, `remove_vertex()`, and `run_single()`.

### 5.7.1.4 pop()

```
void* pop (
    struct stack * stack )
```

References `stack_node::data`, `stack::length`, `stack_node::next`, and `stack::root`.

Referenced by `destroy_graph_avl()`, `destroy_graph_stack()`, `fire()`, `print_edges()`, `print_state()`, `process_requests()`, `remove_vertex()`, `run()`, `run_single()`, and `start_set()`.

### 5.7.1.5 push()

```
int push (
    struct stack * stack,
    void * data )
```

References `stack_node::data`, `stack::length`, `stack_node::next`, and `stack::root`.

Referenced by `inorder_nodes()`, `postorder_nodes()`, `preorder_nodes()`, `stackify_nodes()`, `start_set()`, and `submit_request()`.

## 5.8 include/topologic.h File Reference

```
#include "header.h"
#include "stack.h"
#include "AVL.h"
#include "edge.h"
#include "vertex.h"
#include "request.h"
#include "graph.h"
```

## Data Structures

- struct [fireable](#)

## Macros

- #define [PTHREAD\\_SLEEP\\_TIME](#) 50
- #define [\\_GNU\\_SOURCE](#)
- #define [TOPOLOGIC\\_DEBUG](#) 0
- #define [topologic\\_debug](#)(fmt, ...)
- #define [MAX\\_LOOPS](#) 100
- #define [GRAPH\\_INIT](#)() [graph\\_init](#)(-1, [START\\_STOP](#), [MAX\\_LOOPS](#), [VERTICES](#) | [EDGES](#) | [FUNCTIONS](#) | [GLOBALS](#), [SINGLE](#), [CONTINUE](#))
- #define [CREATE\\_VERTEX](#)(graph, f, id) [create\\_vertex](#)(graph, f, id, NULL, [PROTECT\\_B\\_VARS](#))
- #define [CREATE\\_VERTEX\\_GLBL](#)(graph, f, id, glbl) [create\\_vertex](#)(graph, f, id, glbl, [PROTECT\\_B\\_VARS](#))
- #define [CREATE\\_EDGE](#)(a, b, f) [create\\_edge](#)(a, b, f, NULL)
- #define [CREATE\\_NULL\\_BI\\_EDGE](#)(a, b, f) [create\\_bi\\_edge](#)(a, b, f, NULL, NULL, NULL)
- #define [CREATE\\_BI\\_EDGE](#)(a, b, f, a\_to\_b, b\_to\_a) [create\\_bi\\_edge](#)(a, b, f, NULL, a\_to\_b, b\_to\_a)
- #define [MODIFY\\_VERTEX](#)(vertex, f) [modify\\_vertex](#)(vertex, f, NULL)
- #define [MODIFY\\_VERTEX\\_GLOBALS](#)(vertex, glbl) [modify\\_vertex](#)(vertex, NULL, glbl)
- #define [MODIFY\\_EDGE](#)(a, b, f) [modify\\_edge](#)(a, b, f, NULL)
- #define [MODIFY\\_EDGE\\_GLOBALS](#)(a, b, glbl) [modify\\_edge](#)(a, b, NULL, glbl)
- #define [MODIFY\\_BI\\_EDGE](#)(a, b, f) [modify\\_bi\\_edge](#)(a, b, f, NULL)
- #define [MODIFY\\_BI\\_EDGE\\_GLOBALS](#)(a, b, glbl) [modify\\_bi\\_edge](#)(a, b, NULL, glbl)
- #define [CREATE\\_REQUEST](#)(request, args) [create\\_request](#)(request, args, NULL)

## Functions

- void \* [fire\\_pthread](#) (void \*vargp)
- struct [graph](#) \* [graph\\_init](#) (int max\_state\_changes, int snapshot\_timestamp, int max\_loop, unsigned int lvl ↵ verbose, enum [CONTEXT](#) context, enum [MEM\\_OPTION](#) mem\_option)
- struct [vertex](#) \* [create\\_vertex](#) (struct [graph](#) \*graph, void(\*f)(struct [graph](#) \*, struct [vertex\\_result](#) \*, void \*, void \*), int id, void \*glbl)
- struct [edge](#) \* [create\\_edge](#) (struct [vertex](#) \*a, struct [vertex](#) \*b, int(\*f)(void \*, void \*, const void \*const, const void \*const), void \*glbl)
- int [create\\_bi\\_edge](#) (struct [vertex](#) \*a, struct [vertex](#) \*b, int(\*f)(void \*, void \*, const void \*const, const void \*const), void \*glbl, struct [edge](#) \*\*edge\_a\_to\_b, struct [edge](#) \*\*edge\_b\_to\_a)
- int [remove\\_edge](#) (struct [vertex](#) \*a, struct [vertex](#) \*b)
- int [remove\\_edge\\_id](#) (struct [vertex](#) \*a, int id)
- int [remove\\_bi\\_edge](#) (struct [vertex](#) \*a, struct [vertex](#) \*b)
- int [remove\\_vertex](#) (struct [graph](#) \*graph, struct [vertex](#) \*vertex)
- int [remove\\_vertex\\_id](#) (struct [graph](#) \*graph, int id)
- int [modify\\_vertex](#) (struct [vertex](#) \*vertex, void(\*f)(struct [graph](#) \*, struct [vertex\\_result](#) \*, void \*, void \*), void \*glbl)
- int [modify\\_shared\\_edge\\_vars](#) (struct [vertex](#) \*vertex, void \*edge\_vars)
- int [modify\\_edge](#) (struct [vertex](#) \*a, struct [vertex](#) \*b, int(\*f)(void \*, void \*, const void \*const, const void \*const), void \*glbl)
- int [modify\\_bi\\_edge](#) (struct [vertex](#) \*a, struct [vertex](#) \*b, int(\*f)(void \*, void \*, const void \*const, const void \*const), void \*glbl)
- int [fire](#) (struct [graph](#) \*graph, struct [vertex](#) \*vertex, struct [vertex\\_result](#) \*args, enum [STATES](#) color, int iloop)
- int [switch\\_vertex](#) (struct [graph](#) \*graph, struct [vertex](#) \*vertex, struct [vertex\\_result](#) \*args, enum [STATES](#) color, int iloop)
- int [start\\_set](#) (struct [graph](#) \*graph, int id[], int num\_vertices)

- int [submit\\_request](#) (struct [graph](#) \*, struct [request](#) \*[request](#))
- struct [request](#) \* [create\\_request](#) (enum [REQUESTS](#) [request](#), void \*args, void(\*f)(void \*))
- int [process\\_requests](#) (struct [graph](#) \*[graph](#))
- int [run](#) (struct [graph](#) \*[graph](#), struct [vertex\\_result](#) \*\*[vertex\\_args](#))
- int [resume\\_graph](#) (struct [graph](#) \*[graph](#))
- int [pause\\_graph](#) (struct [graph](#) \*[graph](#))
- void [print\\_graph](#) (struct [graph](#) \*[graph](#))
- struct [graph](#) \* [parse\\_json](#) (const char \*path)
- int [destroy\\_graph](#) (struct [graph](#) \*[graph](#))
- int [destroy\\_request](#) (struct [request](#) \*[request](#))

## 5.8.1 Macro Definition Documentation

### 5.8.1.1 `_GNU_SOURCE`

```
#define _GNU_SOURCE
```

### 5.8.1.2 `CREATE_BI_EDGE`

```
#define CREATE_BI_EDGE(  
    a,  
    b,  
    f,  
    a_to_b,  
    b_to_a ) create\_bi\_edge(a, b, f, NULL, a_to_b, b_to_a)
```

### 5.8.1.3 `CREATE_EDGE`

```
#define CREATE_EDGE(  
    a,  
    b,  
    f ) create\_edge(a, b, f, NULL)
```

### 5.8.1.4 `CREATE_NULL_BI_EDGE`

```
#define CREATE_NULL_BI_EDGE(  
    a,  
    b,  
    f ) create\_bi\_edge(a, b, f, NULL, NULL, NULL)
```

### 5.8.1.5 CREATE\_REQUEST

```
#define CREATE_REQUEST(  
    request,  
    args ) create_request(request, args, NULL)
```

### 5.8.1.6 CREATE\_VERTEX

```
#define CREATE_VERTEX(  
    graph,  
    f,  
    id ) create_vertex(graph, f, id, NULL, PROTECT_B_VARS)
```

### 5.8.1.7 CREATE\_VERTEX\_GLBL

```
#define CREATE_VERTEX_GLBL(  
    graph,  
    f,  
    id,  
    glbl ) create_vertex(graph, f, id, glbl, PROTECT_B_VARS)
```

### 5.8.1.8 GRAPH\_INIT

```
#define GRAPH_INIT( ) graph_init(-1, START_STOP, MAX_LOOPS, VERTICES | EDGES | FUNCTIONS | GLOBALS,  
    SINGLE, CONTINUE)
```

### 5.8.1.9 MAX\_LOOPS

```
#define MAX_LOOPS 100
```

### 5.8.1.10 MODIFY\_BI\_EDGE

```
#define MODIFY_BI_EDGE(  
    a,  
    b,  
    f ) modify_bi_edge(a, b, f, NULL)
```

#### 5.8.1.11 MODIFY\_BI\_EDGE\_GLOBALS

```
#define MODIFY_BI_EDGE_GLOBALS(  
    a,  
    b,  
    glbl ) modify_bi_edge(a, b, NULL, glbl)
```

#### 5.8.1.12 MODIFY\_EDGE

```
#define MODIFY_EDGE(  
    a,  
    b,  
    f ) modify_edge(a, b, f, NULL)
```

#### 5.8.1.13 MODIFY\_EDGE\_GLOBALS

```
#define MODIFY_EDGE_GLOBALS(  
    a,  
    b,  
    glbl ) modify_edge(a, b, NULL, glbl)
```

#### 5.8.1.14 MODIFY\_VERTEX

```
#define MODIFY_VERTEX(  
    vertex,  
    f ) modify_vertex(vertex, f, NULL)
```

#### 5.8.1.15 MODIFY\_VERTEX\_GLOBALS

```
#define MODIFY_VERTEX_GLOBALS(  
    vertex,  
    glbl ) modify_vertex(vertex, NULL, glbl)
```

#### 5.8.1.16 PTHREAD\_SLEEP\_TIME

```
#define PTHREAD_SLEEP_TIME 50
```



### 5.8.1.17 TOPOLOGIC\_DEBUG

```
#define TOPOLOGIC_DEBUG 0
```

### 5.8.1.18 topologic\_debug

```
#define topologic_debug(  
    fmt,  
    ... )
```

**Value:**

```
"%s:%s:%s:%d:%s(): " fmt "\n", __DATE__, __TIME__, \
__VA_ARGS__); } while (0)

do {if (TOPOLOGIC_DEBUG) fprintf(stderr,  
    __FILE__, __LINE__, __func__,
```

## 5.8.2 Function Documentation

### 5.8.2.1 create\_bi\_edge()

```
int create_bi_edge (  
    struct vertex * a,  
    struct vertex * b,  
    int(*) (void *, void *, const void *const, const void *const) f,  
    void * glbl,  
    struct edge ** edge_a_to_b,  
    struct edge ** edge_b_to_a )
```

**@RETURNS** 0 for success; -1 for fail See create\_edge Will create an bidirectional edge between vertex a and b with some criteria determined by the function f. Will store the edges in edge\_a and edge\_b. If edge\_a\_to\_b or edge\_b\_to\_a is NULL it will not.

**References** edge::a, edge::b, BI\_EDGE, edge::bi\_edge, edge::bi\_edge\_lock, vertex::context, create\_edge(), edge::edge\_type, edge::f, edge::glbl, remove\_edge(), SINGLE, and topologic\_debug.

**Referenced by** procces\_request().

### 5.8.2.2 create\_edge()

```
struct edge* create_edge (  
    struct vertex * a,  
    struct vertex * b,  
    int(*) (void *, void *, const void *const, const void *const) f,  
    void * glbl )
```

**@PARAM** a: A vertex **@PARAM** b: Another vertex (can be 'a') **@PARAM** f: a function **@PARAM** glbl: global variables **@RETURN** the edge connecting a to b Will create an edge from vertex a to b with some criteria determined by the function f. NOTE: NULL glbl will mean no global variables. f cannot be NULL.

**References** edge::a, edge::a\_vars, edge::b, edge::b\_vars, edge::bi\_edge, vertex::context, EDGE, shared\_edge↔::edge\_data, vertex::edge\_tree, edge::edge\_type, edge::f, find(), edge::glbl, edge::id, vertex::id, insert(), vertex↔::joining\_vertices, vertex::lock, remove\_edge(), remove\_ID(), SELF\_EDGE, vertex::shared, SINGLE, SWITCH, and topologic\_debug.

**Referenced by** create\_bi\_edge(), and procces\_request().

### 5.8.2.3 create\_request()

```
struct request* create_request (
    enum REQUESTS request,
    void * args,
    void(*) (void *) f )
```

@PARAM request: the desired request @PARAM args: the arguments needed for f @PARAM f: the function of the request @RETURN the request or NULL if it fails Creates a request structure to be called later

References request::args, CREAT\_BI\_EDGE, CREAT\_EDGE, CREAT\_VERTEX, DESTROY\_BI\_EDGE, DESTROY\_EDGE, DESTROY\_EDGE\_BY\_ID, DESTROY\_VERTEX, DESTROY\_VERTEX\_BY\_ID, request::f, GENERIC, MOD\_BI\_EDGE, MOD\_EDGE, MOD\_EDGE\_VARS, MOD\_VERTEX, request::request, and topologic↵\_debug.

### 5.8.2.4 create\_vertex()

```
struct vertex* create_vertex (
    struct graph * graph,
    void(*) (struct graph *, struct vertex_result *, void *, void *) f,
    int id,
    void * glbl )
```

@PARAM graph: the graph @PARAM f: a function @PARAM id: vertex id to be used: MUST BE UNIQUE @PARAM glbl: global variables @RETURN vertex: a vertex to be used in a graph On creation a process will be spawned for the vertex The vertex will compute function f when called NOTE: NULL glbl will mean no global variables. f cannot be NULL.

References graph::context, vertex::context, destroy\_avl(), vertex::edge\_tree, vertex::f, vertex::glbl, vertex::id, init\_avl(), insert(), vertex::is\_active, vertex::joining\_vertices, graph::lock, vertex::lock, vertex::shared, SINGLE, topologic\_debug, shared\_edge::vertex\_data, and graph::vertices.

Referenced by procces\_request().

### 5.8.2.5 destroy\_graph()

```
int destroy_graph (
    struct graph * graph )
```

@PARAM graph: the graph @RETURN -1 for fail; 0 for success Destroys and frees the graph

References graph::black\_fire, graph::black\_locked, graph::black\_vertex\_count, graph::color\_lock, graph::context, destroy\_graph\_avl(), destroy\_graph\_stack(), graph::lock, graph::modify, graph::pause\_cond, graph::red\_fire, graph::red\_locked, graph::red\_vertex\_count, graph::remove\_edges, graph::remove\_vertices, SINGLE, graph↵::start, graph::state, TERMINATE, topologic\_debug, and graph::vertices.

### 5.8.2.6 destroy\_request()

```
int destroy_request (
    struct request * request )
```

@PARAM request: a request @RETURN -1 for fail; 0 for success Destroys and frees a request

References request::args, request::f, request::request, and topologic\_debug.

Referenced by destroy\_graph\_stack().

### 5.8.2.7 fire()

```
int fire (
    struct graph * graph,
    struct vertex * vertex,
    struct vertex_result * args,
    enum STATES color,
    int iloop )
```

@PARAM graph: the graph @PARAM vertex: A vertex to be ran @PARAM args: arguments @PARAM color: the state in which unlocks fire process if STATE is set to PRINT then fire will fail @PARAM iloop: the number of times that vertex fired in succession @RETURNS the result of the vertex fire will wake up the vertex and pass args to the vertex to compute its function and then call switch and clean itself up

References edge::a\_vars, fireable::args, edge::b, edge::b\_vars, BI\_EDGE, edge::bi\_edge\_lock, BLACK, graph↔::black\_fire, graph::black\_locked, graph::black\_vertex\_count, fireable::color, graph::context, destroy\_stack(), vertex\_result::edge\_argv, vertex::edge\_tree, edge::edge\_type, vertex::f, edge::f, fire(), edge::glbl, vertex::glbl, fireable::iloop, init\_stack(), vertex::is\_active, graph::lock, vertex::lock, graph::max\_loop, graph::max\_state\_changes, NONE, graph::num\_vertices, pop(), preorder(), PTHREAD\_SLEEP\_TIME, RED, graph::red\_fire, graph::red\_locked, graph::red\_vertex\_count, vertex::shared, sleep\_ms(), graph::state, graph::state\_count, SWITCH, SWITCH↔ UNSAFE, switch\_vertex(), TERMINATE, topologic\_debug, vertex\_result::vertex\_argv, and shared\_edge::vertex↔ data.

Referenced by fire(), and fire\_pthread().

### 5.8.2.8 fire\_pthread()

```
void* fire_pthread (
    void * vargp )
```

Wrapper function for fire, fire\_1 @PARAM vargp: arguments

References fireable::args, fireable::color, fire(), fireable::graph, fireable::iloop, PTHREAD\_SLEEP\_TIME, sleep↔ ms(), topologic\_debug, and fireable::vertex.

Referenced by run(), and switch\_vertex().

### 5.8.2.9 graph\_init()

```
struct graph* graph_init (
    int max_state_changes,
    int snapshot_timestamp,
    int max_loop,
    unsigned int lvl_verbose,
    enum CONTEXT context,
    enum MEM_OPTION mem_option )
```

@PARAM max\_state\_changes: # state changes before entering sink vertex due to infinite loop of states; -1 to ignore @PARAM snapshot\_timestamp: printing out data at given timestamp for user; -1 for none; 0 for first and last state @PARAM lvl\_verbose: how verbose timestamp print is @PARAM context: linear or context-switch based @RETURN an empty graph Creates a graph structures

References graph::black\_fire, graph::black\_locked, graph::black\_vertex\_count, graph::color\_lock, graph::context, destroy\_avl(), destroy\_stack(), init\_avl(), init\_stack(), graph::lock, graph::lvl\_verbose, graph::max\_loop, graph::max\_state\_changes, graph::mem\_option, graph::modify, NONE, graph::num\_vertices, graph::pause, graph::pause\_cond, graph::previous\_color, graph::print\_flag, RED, graph::red\_fire, graph::red\_locked, graph::red\_vertex\_count, graph::remove\_edges, graph::remove\_vertices, SINGLE, graph::snapshot\_timestamp, graph::start, graph::state, graph::state\_count, topologic\_debug, and graph::vertices.

### 5.8.2.10 modify\_bi\_edge()

```
int modify_bi_edge (
    struct vertex * a,
    struct vertex * b,
    int(*) (void *, void *, const void *const, const void *const) f,
    void * glbl )
```

@PARAM a: a vertex @PARAM b: another vertex @PARAM f: a function @PARAM glbl: global variables @RETURN 0 for success; -1 for fail; -2 if only edge from a to b is modified; -3 if only edge from b to a is modified Modifies the edge between a and b's function and variables NOTE: NULL f, or glbl will mean no change.

References edge::a, edge::b, edge::f, edge::glbl, modify\_edge(), and topologic\_debug.

Referenced by procces\_request().

### 5.8.2.11 modify\_edge()

```
int modify_edge (
    struct vertex * a,
    struct vertex * b,
    int(*) (void *, void *, const void *const, const void *const) f,
    void * glbl )
```

@PARAM a: a vertex @PARAM b: another vertex @PARAM f: a function @PARAM glbl: global variables @RETURN 0 for success; -1 for fail Modifies the edge connecting a to b's function NOTE: NULL f, or glbl will mean no change.

References edge::a, edge::b, vertex::context, vertex::edge\_tree, edge::f, find(), edge::glbl, vertex::id, vertex::lock, SINGLE, and topologic\_debug.

Referenced by modify\_bi\_edge(), and procces\_request().

**5.8.2.12 modify\_shared\_edge\_vars()**

```
int modify_shared_edge_vars (
    struct vertex * vertex,
    void * edge_vars )
```

@PARAM vertex: a vertex @PARAM edge\_vars: shared variables @RETURN 0 for success; -1 for fail Modifies the vertices shared variables with it's edges

References vertex::context, vertex::lock, vertex::shared, SINGLE, topologic\_debug, and shared\_edge::vertex\_data.

Referenced by procces\_request().

**5.8.2.13 modify\_vertex()**

```
int modify_vertex (
    struct vertex * vertex,
    void(*) (struct graph *, struct vertex_result *, void *, void *) f,
    void * glbl )
```

@PARAM vertex: a vertex @PARAM f: a function @PARAM glbl: global variables NOTE: NULL f, or glbl will mean no change. @RETURN 0 for success; -1 for fail Modifies the vertices function

References vertex::context, vertex::f, vertex::glbl, vertex::lock, SINGLE, and topologic\_debug.

Referenced by procces\_request().

**5.8.2.14 parse\_json()**

```
struct graph* parse_json (
    const char * path )
```

@PARAM path: path to graph input file @RETURN the graph specified in the file

**5.8.2.15 pause\_graph()**

```
int pause_graph (
    struct graph * graph )
```

@PARAM graph: the graph @RETURN 0 for success -1 if it fails Pauses run

References graph::lock, graph::pause, graph::pause\_cond, and topologic\_debug.

### 5.8.2.16 print\_graph()

```
void print_graph (
    struct graph * graph )
```

@PARAM graph: the graph Prints the graph with desired output TODO: Print enums

References graph::context, graph::lock, graph::lvl\_verbose, graph::max\_loop, graph::max\_state\_changes, NO↵\_VERB, graph::num\_vertices, print\_state(), SINGLE, AVLTree::size, graph::snapshot\_timestamp, START\_STOP, graph::state\_count, topologic\_debug, and graph::vertices.

Referenced by run(), and run\_single().

### 5.8.2.17 process\_requests()

```
int process_requests (
    struct graph * graph )
```

@PARAM graph: the graph @RETURN 0 if all got processed; -1 if a request failed will set ERRNO to the ENUM Process requests that are queued in the graph

References graph::context, graph::lock, graph::modify, pop(), procces\_request(), graph::remove\_edges, graph↵::remove\_vertices, SINGLE, and topologic\_debug.

Referenced by run(), and run\_single().

### 5.8.2.18 remove\_bi\_edge()

```
int remove_bi_edge (
    struct vertex * a,
    struct vertex * b )
```

@PARAM a: a vertex @PARAM b: another vertex @RETURN 0 for success; -1 for fail; -2 if only edge from a to b is removed; -3 if only edge from b to a is removed Removes the edge connecting a to b

References edge::a, edge::b, remove\_edge(), and topologic\_debug.

Referenced by procces\_request().

### 5.8.2.19 remove\_edge()

```
int remove_edge (
    struct vertex * a,
    struct vertex * b )
```

@PARAM a: a vertex @PARAM b: another vertex @RETURN 0 for success; -1 for fail Removes the edge connecting a to b

References edge::a, edge::a\_vars, edge::b, edge::b\_vars, BI\_EDGE, edge::bi\_edge, edge::bi\_edge\_lock, vertex↵::context, EDGE, vertex::edge\_tree, edge::edge\_type, edge::f, edge::glbl, edge::id, vertex::id, vertex::joining↵\_vertices, vertex::lock, remove\_ID(), SINGLE, and topologic\_debug.

Referenced by create\_bi\_edge(), create\_edge(), procces\_request(), and remove\_bi\_edge().

**5.8.2.20 remove\_edge\_id()**

```
int remove_edge_id (
    struct vertex * a,
    int id )
```

@PARAM a: a vertex @PARAM id: id of edge to remove @RETURN 0 for success; -1 for fail Removes the edge in a with that id

References edge::a, edge::a\_vars, edge::b, edge::b\_vars, BI\_EDGE, edge::bi\_edge, edge::bi\_edge\_lock, vertex::context, EDGE, vertex::edge\_tree, edge::edge\_type, edge::f, edge::glbl, edge::id, vertex::id, vertex::joining\_vertices, vertex::lock, remove\_ID(), SINGLE, and topologic\_debug.

Referenced by procces\_request(), and remove\_vertex().

**5.8.2.21 remove\_vertex()**

```
int remove_vertex (
    struct graph * graph,
    struct vertex * vertex )
```

@PARAM graph: the graph @PARAM vertex: a vertex @RETURN 0 for success; -1 for fail Removes the vertex and all connected edges

References edge::a, edge::b, BI\_EDGE, edge::bi\_edge, edge::bi\_edge\_lock, graph::context, vertex::context, destroy\_stack(), EDGE, vertex::edge\_tree, edge::edge\_type, edge::f, edge::glbl, vertex::glbl, edge::id, vertex::id, init\_stack(), vertex::joining\_vertices, graph::lock, vertex::lock, pop(), remove\_edge\_id(), remove\_ID(), SELF\_EDGE, vertex::shared, SINGLE, stackify(), topologic\_debug, shared\_edge::vertex\_data, and graph::vertices.

Referenced by destroy\_graph\_avl(), procces\_request(), and remove\_vertex\_id().

**5.8.2.22 remove\_vertex\_id()**

```
int remove_vertex_id (
    struct graph * graph,
    int id )
```

@PARAM graph: the graph @PARAM id: the vertex id @RETURN 0 for success; -1 for fail Removes the vertex and all connected edges

References graph::context, find(), graph::lock, remove\_vertex(), SINGLE, topologic\_debug, and graph::vertices.

Referenced by procces\_request().

### 5.8.2.23 resume\_graph()

```
int resume_graph (
    struct graph * graph )
```

@PARAM graph: the graph @RETURN 0 for success -1 if it fails Resumes run

References graph::lock, graph::pause, and topologic\_debug.

### 5.8.2.24 run()

```
int run (
    struct graph * graph,
    struct vertex_result ** vertex_args )
```

@PARAM graph: the graph @PARAM vertex\_args: array of vertex arguments for f @RETURN 0 if run terminates normally -1 if it fails Attempts to run the graph else aborts.

References ABORT, fireable::args, BLACK, graph::black\_fire, graph::black\_locked, graph::black\_vertex\_count, fireable::color, graph::color\_lock, graph::context, CONTINUE, fire\_pthread(), fireable::graph, fireable::iloop, vertex↵::is\_active, graph::lock, MAX\_ATTEMPTS, graph::max\_state\_changes, graph::mem\_option, graph::num\_vertices, graph::pause, graph::pause\_cond, pop(), graph::previous\_color, PRINT, graph::print\_flag, print\_graph(), process↵\_requests(), RED, graph::red\_fire, graph::red\_locked, graph::red\_vertex\_count, run\_single(), SINGLE, graph::start, graph::state, graph::state\_count, TERMINATE, THREAD\_ATTEMPT\_SLEEP, topologic\_debug, fireable::vertex, and WAIT.

### 5.8.2.25 start\_set()

```
int start_set (
    struct graph * graph,
    int id[],
    int num_vertices )
```

@PARAM graph: the graph, @PARAM id: the ids of the vertices @PARAM num\_vertices: number of vertices @RETURN -1 for fail if any vertex fails; 0 for success Creates multiple contexts that are ran in parallel Handle errors

Given vertx failed, so at this point, free the vertices and leave

References graph::context, find(), vertex::id, pop(), push(), SINGLE, graph::start, topologic\_debug, and graph↵::vertices.



### 5.8.2.26 submit\_request()

```
int submit_request (
    struct graph * graph,
    struct request * request )
```

@PARAM graph: the graph @PARAM request: the request to be processed @RETRUN -1 for fail; 0 for succes; Submits a request to be processed after all active nodes complete

References graph::context, DESTROY\_BI\_EDGE, DESTROY\_EDGE, DESTROY\_EDGE\_BY\_ID, DESTROY\_VERTEX, DESTROY\_VERTEX\_BY\_ID, graph::lock, graph::modify, push(), graph::remove\_edges, graph::remove\_vertices, request::request, SINGLE, and topologic\_debug.

### 5.8.2.27 switch\_vertex()

```
int switch_vertex (
    struct graph * graph,
    struct vertex * vertex,
    struct vertex_result * args,
    enum STATES color,
    int iloop )
```

@PARAM graph: the graph @PARAM vertex: The vertex in which just fire @PARAM args; The result of the vertex Upon call the switch function will compute the edge functions connected to the vertex @PARAM iloop: the number of times that vertex fired in succession @RETURNS 0 On success; the vertex connected to the successful edge will be fired; -1 on failure

References ABORT, fireable::args, fireable::color, CONTINUE, vertex\_result::edge\_argv, vertex\_result::edge\_size, fire\_thread(), fireable::graph, fireable::iloop, MAX\_ATTEMPTS, graph::mem\_option, THREAD\_ATTEMPT\_SLEEP, topologic\_debug, fireable::vertex, vertex\_result::vertex\_argv, vertex\_result::vertex\_size, and WAIT.

Referenced by fire().

## 5.9 include/vertex.h File Reference

```
#include <signal.h>
#include <stdio.h>
#include "../stack.h"
#include "../AVL.h"
#include "../context.h"
#include "../graph.h"
```

### Data Structures

- struct [vertex\\_result](#)
- union [shared\\_edge](#)
- struct [vertex](#)
- struct [vertex\\_request](#)
- struct [mod\\_vertex\\_request](#)
- struct [mod\\_edge\\_vars\\_request](#)
- struct [destroy\\_vertex\\_request](#)
- struct [destroy\\_vertex\\_id\\_request](#)

## 5.10 README.md File Reference

## 5.11 src/AVL.c File Reference

```
#include "../include/AVL.h"
```

### Functions

- struct [AVLTree](#) \* [init\\_avl](#) ()
- int [balance](#) (struct [AVLNode](#) \*node)
- int [max\\_height](#) (struct [AVLNode](#) \*node)
- struct [AVLNode](#) \* [right\\_rotate](#) (struct [AVLNode](#) \*node)
- struct [AVLNode](#) \* [left\\_rotate](#) (struct [AVLNode](#) \*node)
- struct [AVLNode](#) \* [create\\_node](#) (void \*data, int id)
- struct [AVLNode](#) \* [insert\\_node](#) (struct [AVLNode](#) \*node, struct [AVLNode](#) \*insert)
- int [insert](#) (struct [AVLTree](#) \*tree, void \*data, int id)
- void \* [find\\_node](#) (struct [AVLNode](#) \*root, int id)
- void \* [find](#) (struct [AVLTree](#) \*tree, int id)
- struct [AVLNode](#) \* [minNode](#) (struct [AVLNode](#) \*node)
- struct [AVLNode](#) \* [remove\\_node](#) (struct [AVLNode](#) \*root, int id, void \*\*data)
- void \* [remove\\_ID](#) (struct [AVLTree](#) \*tree, int id)
- void [inorder\\_nodes](#) (struct [AVLNode](#) \*node, struct [stack](#) \*stack)
- void [inorder](#) (struct [AVLTree](#) \*tree, struct [stack](#) \*stack)
- void [preorder\\_nodes](#) (struct [AVLNode](#) \*node, struct [stack](#) \*stack)
- void [preorder](#) (struct [AVLTree](#) \*tree, struct [stack](#) \*stack)
- void [postorder\\_nodes](#) (struct [AVLNode](#) \*node, struct [stack](#) \*stack)
- void [postorder](#) (struct [AVLTree](#) \*tree, struct [stack](#) \*stack)
- void [stackify\\_nodes](#) (struct [AVLNode](#) \*node, struct [stack](#) \*stack)
- void [stackify](#) (struct [AVLTree](#) \*tree, struct [stack](#) \*stack)
- void [destroy\\_avl\\_nodes](#) (struct [AVLNode](#) \*node)
- void [destroy\\_avl](#) (struct [AVLTree](#) \*tree)

### 5.11.1 Function Documentation

#### 5.11.1.1 balance()

```
int balance (
    struct AVLNode * node )
```

References [AVLNode::height](#), [AVLNode::left](#), and [AVLNode::right](#).

Referenced by [insert\\_node\(\)](#), and [remove\\_node\(\)](#).

### 5.11.1.2 create\_node()

```
struct AVLNode* create_node (
    void * data,
    int id )
```

References AVLNode::data, AVLNode::height, AVLNode::id, AVLNode::left, and AVLNode::right.

Referenced by insert().

### 5.11.1.3 destroy\_avl()

```
void destroy_avl (
    struct AVLTree * tree )
```

References destroy\_avl\_nodes(), AVLTree::root, and AVLTree::size.

Referenced by create\_vertex(), destroy\_graph\_avl(), and graph\_init().

### 5.11.1.4 destroy\_avl\_nodes()

```
void destroy_avl_nodes (
    struct AVLNode * node )
```

References AVLNode::data, AVLNode::height, AVLNode::left, and AVLNode::right.

Referenced by destroy\_avl().

### 5.11.1.5 find()

```
void* find (
    struct AVLTree * tree,
    int id )
```

References find\_node(), and AVLTree::root.

Referenced by create\_edge(), modify\_edge(), remove\_vertex\_id(), and start\_set().

#### 5.11.1.6 find\_node()

```
void* find_node (
    struct AVLNode * root,
    int id )
```

References AVLNode::data, AVLNode::id, AVLNode::left, and AVLNode::right.

Referenced by find().

#### 5.11.1.7 init\_avl()

```
struct AVLTree* init_avl ( )
```

References AVLTree::root, and AVLTree::size.

Referenced by create\_vertex(), and graph\_init().

#### 5.11.1.8 inorder()

```
void inorder (
    struct AVLTree * tree,
    struct stack * stack )
```

References inorder\_nodes(), AVLTree::root, and AVLTree::size.

Referenced by print\_state().

#### 5.11.1.9 inorder\_nodes()

```
void inorder_nodes (
    struct AVLNode * node,
    struct stack * stack )
```

References AVLNode::data, AVLNode::id, AVLNode::left, push(), and AVLNode::right.

Referenced by inorder().

#### 5.11.1.10 insert()

```
int insert (
    struct AVLTree * tree,
    void * data,
    int id )
```

References `create_node()`, `AVLNode::data`, `insert_node()`, `AVLTree::root`, and `AVLTree::size`.

Referenced by `create_edge()`, `create_vertex()`, and `insert_node()`.

#### 5.11.1.11 insert\_node()

```
struct AVLNode* insert_node (
    struct AVLNode * node,
    struct AVLNode * insert )
```

References `balance()`, `AVLNode::height`, `AVLNode::id`, `insert()`, `AVLNode::left`, `left_rotate()`, `max_height()`, `AVLNode::right`, and `right_rotate()`.

Referenced by `insert()`.

#### 5.11.1.12 left\_rotate()

```
struct AVLNode* left_rotate (
    struct AVLNode * node )
```

References `AVLNode::height`, `AVLNode::left`, `max_height()`, and `AVLNode::right`.

Referenced by `insert_node()`, and `remove_node()`.

#### 5.11.1.13 max\_height()

```
int max_height (
    struct AVLNode * node )
```

References `AVLNode::height`, `AVLNode::left`, and `AVLNode::right`.

Referenced by `insert_node()`, `left_rotate()`, `remove_node()`, and `right_rotate()`.

#### 5.11.1.14 minNode()

```
struct AVLNode* minNode (
    struct AVLNode * node )
```

References AVLNode::left.

Referenced by remove\_node().

#### 5.11.1.15 postorder()

```
void postorder (
    struct AVLTree * tree,
    struct stack * stack )
```

References postorder\_nodes(), AVLTree::root, and AVLTree::size.

#### 5.11.1.16 postorder\_nodes()

```
void postorder_nodes (
    struct AVLNode * node,
    struct stack * stack )
```

References AVLNode::data, AVLNode::id, AVLNode::left, push(), and AVLNode::right.

Referenced by postorder().

#### 5.11.1.17 preorder()

```
void preorder (
    struct AVLTree * tree,
    struct stack * stack )
```

References preorder\_nodes(), AVLTree::root, and AVLTree::size.

Referenced by destroy\_graph\_avl(), fire(), print\_edges(), and run\_single().

#### 5.11.1.18 preorder\_nodes()

```
void preorder_nodes (
    struct AVLNode * node,
    struct stack * stack )
```

References AVLNode::data, AVLNode::id, AVLNode::left, push(), and AVLNode::right.

Referenced by preorder().

#### 5.11.1.19 remove\_ID()

```
void* remove_ID (
    struct AVLTree * tree,
    int id )
```

References AVLNode::data, remove\_node(), AVLTree::root, and AVLTree::size.

Referenced by create\_edge(), remove\_edge(), remove\_edge\_id(), and remove\_vertex().

#### 5.11.1.20 remove\_node()

```
struct AVLNode* remove_node (
    struct AVLNode * root,
    int id,
    void ** data )
```

References balance(), AVLNode::data, AVLNode::height, AVLNode::id, AVLNode::left, left\_rotate(), max\_height(), minNode(), AVLNode::right, and right\_rotate().

Referenced by remove\_ID().

#### 5.11.1.21 right\_rotate()

```
struct AVLNode* right_rotate (
    struct AVLNode * node )
```

References AVLNode::height, AVLNode::left, max\_height(), and AVLNode::right.

Referenced by insert\_node(), and remove\_node().

#### 5.11.1.22 stackify()

```
void stackify (
    struct AVLTree * tree,
    struct stack * stack )
```

References AVLTree::root, AVLTree::size, and stackify\_nodes().

Referenced by remove\_vertex().

### 5.11.1.23 stackify\_nodes()

```
void stackify_nodes (
    struct AVLNode * node,
    struct stack * stack )
```

References AVLNode::data, AVLNode::height, AVLNode::left, push(), and AVLNode::right.

Referenced by stackify().

## 5.12 src/edge.c File Reference

```
#include "../include/topologic.h"
```

### Functions

- struct [edge](#) \* [create\\_edge](#) (struct [vertex](#) \*a, struct [vertex](#) \*b, int(\*f)(void \*, void \*, const void \*const, const void \*const), void \*gbl)
- int [create\\_bi\\_edge](#) (struct [vertex](#) \*a, struct [vertex](#) \*b, int(\*f)(void \*, void \*, const void \*const, const void \*const), void \*gbl, struct [edge](#) \*\*edge\_a\_to\_b, struct [edge](#) \*\*edge\_b\_to\_a)
- int [remove\\_edge](#) (struct [vertex](#) \*a, struct [vertex](#) \*b)
- int [remove\\_edge\\_id](#) (struct [vertex](#) \*a, int id)
- int [remove\\_bi\\_edge](#) (struct [vertex](#) \*a, struct [vertex](#) \*b)
- int [modify\\_edge](#) (struct [vertex](#) \*a, struct [vertex](#) \*b, int(\*f)(void \*, void \*, const void \*const, const void \*const), void \*gbl)
- int [modify\\_bi\\_edge](#) (struct [vertex](#) \*a, struct [vertex](#) \*b, int(\*f)(void \*, void \*, const void \*const, const void \*const), void \*gbl)

### 5.12.1 Function Documentation

#### 5.12.1.1 create\_bi\_edge()

```
int create_bi_edge (
    struct vertex * a,
    struct vertex * b,
    int(*) (void *, void *, const void *const, const void *const) f,
    void * gbl,
    struct edge ** edge_a_to_b,
    struct edge ** edge_b_to_a )
```

@RETURNS 0 for success; -1 for fail See [create\\_edge](#) Will create an bidirectional edge between vertex a and b with some criteria determined by the function f. Will store the edges in edge\_a and edge\_b. If edge\_a\_to\_b or edge\_b\_to\_a is NULL it will not.

References [edge::a](#), [edge::b](#), [BI\\_EDGE](#), [edge::bi\\_edge](#), [edge::bi\\_edge\\_lock](#), [vertex::context](#), [create\\_edge\(\)](#), [edge::edge\\_type](#), [edge::f](#), [edge::gbl](#), [remove\\_edge\(\)](#), [SINGLE](#), and [topologic\\_debug](#).

Referenced by [procces\\_request\(\)](#).



### 5.12.1.2 create\_edge()

```
struct edge* create_edge (
    struct vertex * a,
    struct vertex * b,
    int(*) (void *, void *, const void *const, const void *const) f,
    void * glbl )
```

@PARAM a: A vertex @PARAM b: Another vertex (can be 'a') @PARAM f: a function @PARAM glbl: global variables @RETURN the edge connecting a to b Will create an edge from vertex a to b with some criteria determined by the function f. NOTE: NULL glbl will mean no global variables. f cannot be NULL.

References edge::a, edge::a\_vars, edge::b, edge::b\_vars, edge::bi\_edge, vertex::context, EDGE, shared\_edge↔::edge\_data, vertex::edge\_tree, edge::edge\_type, edge::f, find(), edge::glbl, edge::id, vertex::id, insert(), vertex↔::joining\_vertices, vertex::lock, remove\_edge(), remove\_ID(), SELF\_EDGE, vertex::shared, SINGLE, SWITCH, and topologic\_debug.

Referenced by create\_bi\_edge(), and procces\_request().

### 5.12.1.3 modify\_bi\_edge()

```
int modify_bi_edge (
    struct vertex * a,
    struct vertex * b,
    int(*) (void *, void *, const void *const, const void *const) f,
    void * glbl )
```

@PARAM a: a vertex @PARAM b: another vertex @PARAM f: a function @PARAM glbl: global variables @↔RETURN 0 for success; -1 for fail; -2 if only edge from a to b is modified; -3 if only edge from b to a is modified Modifies the edge between a and b's function and variables NOTE: NULL f, or glbl will mean no change.

References edge::a, edge::b, edge::f, edge::glbl, modify\_edge(), and topologic\_debug.

Referenced by procces\_request().

### 5.12.1.4 modify\_edge()

```
int modify_edge (
    struct vertex * a,
    struct vertex * b,
    int(*) (void *, void *, const void *const, const void *const) f,
    void * glbl )
```

@PARAM a: a vertex @PARAM b: another vertex @PARAM f: a function @PARAM glbl: global variables @↔RETURN 0 for success; -1 for fail Modifies the edge connecting a to b's function NOTE: NULL f, or glbl will mean no change.

References edge::a, edge::b, vertex::context, vertex::edge\_tree, edge::f, find(), edge::glbl, vertex::id, vertex::lock, SINGLE, and topologic\_debug.

Referenced by modify\_bi\_edge(), and procces\_request().

### 5.12.1.5 remove\_bi\_edge()

```
int remove_bi_edge (
    struct vertex * a,
    struct vertex * b )
```

@PARAM a: a vertex @PARAM b: another vertex @RETURN 0 for success; -1 for fail; -2 if only edge from a to b is removed; -3 if only edge from b to a is removed Removes the edge connecting a to b

References edge::a, edge::b, remove\_edge(), and topologic\_debug.

Referenced by procces\_request().

### 5.12.1.6 remove\_edge()

```
int remove_edge (
    struct vertex * a,
    struct vertex * b )
```

@PARAM a: a vertex @PARAM b: another vertex @RETURN 0 for success; -1 for fail Removes the edge connecting a to b

References edge::a, edge::a\_vars, edge::b, edge::b\_vars, BI\_EDGE, edge::bi\_edge, edge::bi\_edge\_lock, vertex↔::context, EDGE, vertex::edge\_tree, edge::edge\_type, edge::f, edge::glbl, edge::id, vertex::id, vertex::joining↔\_vertices, vertex::lock, remove\_ID(), SINGLE, and topologic\_debug.

Referenced by create\_bi\_edge(), create\_edge(), procces\_request(), and remove\_bi\_edge().

### 5.12.1.7 remove\_edge\_id()

```
int remove_edge_id (
    struct vertex * a,
    int id )
```

@PARAM a: a vertex @PARAM id: id of edge to remove @RETURN 0 for success; -1 for fail Removes the edge in a with that id

References edge::a, edge::a\_vars, edge::b, edge::b\_vars, BI\_EDGE, edge::bi\_edge, edge::bi\_edge\_lock, vertex↔::context, EDGE, vertex::edge\_tree, edge::edge\_type, edge::f, edge::glbl, edge::id, vertex::id, vertex::joining↔\_vertices, vertex::lock, remove\_ID(), SINGLE, and topologic\_debug.

Referenced by procces\_request(), and remove\_vertex().

## 5.13 src/graph.c File Reference

```
#include "../include/topologic.h"
```

## Functions

- struct [graph](#) \* [graph\\_init](#) (int max\_state\_changes, int snapshot\_timestamp, int max\_loop, unsigned int lvl\_↵ verbose, enum [CONTEXT](#) context, enum [MEM\\_OPTION](#) mem\_option)
- void [destroy\\_graph\\_stack](#) (struct [stack](#) \*stack)
- void [destroy\\_graph\\_avl](#) (struct [graph](#) \*graph, struct [AVLTree](#) \*tree)
- int [destroy\\_graph](#) (struct [graph](#) \*graph)

### 5.13.1 Function Documentation

#### 5.13.1.1 [destroy\\_graph\(\)](#)

```
int destroy_graph (
    struct graph * graph )
```

@PARAM graph: the graph @RETURN -1 for fail; 0 for success Destroys and frees the graph

References [graph::black\\_fire](#), [graph::black\\_locked](#), [graph::black\\_vertex\\_count](#), [graph::color\\_lock](#), [graph::context](#), [destroy\\_graph\\_avl\(\)](#), [destroy\\_graph\\_stack\(\)](#), [graph::lock](#), [graph::modify](#), [graph::pause\\_cond](#), [graph::red\\_fire](#), [graph::red\\_locked](#), [graph::red\\_vertex\\_count](#), [graph::remove\\_edges](#), [graph::remove\\_vertices](#), [SINGLE](#), [graph::start](#), [graph::state](#), [TERMINATE](#), [topologic\\_debug](#), and [graph::vertices](#).

#### 5.13.1.2 [destroy\\_graph\\_avl\(\)](#)

```
void destroy_graph_avl (
    struct graph * graph,
    struct AVLTree * tree )
```

References [destroy\\_avl\(\)](#), [destroy\\_stack\(\)](#), [init\\_stack\(\)](#), [pop\(\)](#), [preorder\(\)](#), and [remove\\_vertex\(\)](#).

Referenced by [destroy\\_graph\(\)](#).

#### 5.13.1.3 [destroy\\_graph\\_stack\(\)](#)

```
void destroy_graph_stack (
    struct stack * stack )
```

References [destroy\\_request\(\)](#), [destroy\\_stack\(\)](#), and [pop\(\)](#).

Referenced by [destroy\\_graph\(\)](#).

### 5.13.1.4 graph\_init()

```
struct graph* graph_init (
    int max_state_changes,
    int snapshot_timestamp,
    int max_loop,
    unsigned int lvl_verbose,
    enum CONTEXT context,
    enum MEM_OPTION mem_option )
```

@PARAM max\_state\_changes: # state changes before entering sink vertex due to infinite loop of states; -1 to ignore @PARAM snapshot\_timestamp: printing out data at given timestamp for user; -1 for none; 0 for first and last state @PARAM lvl\_verbose: how verbose timestamp print is @PARAM context: linear or context-switch based @RETURN an empty graph Creates a graph structures

References graph::black\_fire, graph::black\_locked, graph::black\_vertex\_count, graph::color\_lock, graph::context, destroy\_avl(), destroy\_stack(), init\_avl(), init\_stack(), graph::lock, graph::lvl\_verbose, graph::max\_loop, graph::max\_state\_changes, graph::mem\_option, graph::modify, NONE, graph::num\_vertices, graph::pause, graph::pause\_cond, graph::previous\_color, graph::print\_flag, RED, graph::red\_fire, graph::red\_locked, graph::red\_vertex\_count, graph::remove\_edges, graph::remove\_vertices, SINGLE, graph::snapshot\_timestamp, graph::start, graph::state, graph::state\_count, topologic\_debug, and graph::vertices.

## 5.14 src/request.c File Reference

```
#include "../include/topologic.h"
```

### Functions

- struct request \* create\_request (enum REQUESTS request, void \*args, void(\*f)(void \*))
- int submit\_request (struct graph \*graph, struct request \*request)
- int procces\_request (struct request \*request)
- int process\_requests (struct graph \*graph)
- int destroy\_request (struct request \*request)

### 5.14.1 Function Documentation

#### 5.14.1.1 create\_request()

```
struct request* create_request (
    enum REQUESTS request,
    void * args,
    void(*) (void *) f )
```

@PARAM request: the desired request @PARAM args: the arguments needed for f @PARAM f: the function of the request @RETURN the request or NULL if it fails Creates a request structure to be called later

References request::args, CREAT\_BI\_EDGE, CREAT\_EDGE, CREAT\_VERTEX, DESTROY\_BI\_EDGE, DESTROY\_EDGE, DESTROY\_EDGE\_BY\_ID, DESTROY\_VERTEX, DESTROY\_VERTEX\_BY\_ID, request::f, GENERIC, MOD\_BI\_EDGE, MOD\_EDGE, MOD\_EDGE\_VARS, MOD\_VERTEX, request::request, and topologic\_debug.

### 5.14.1.2 destroy\_request()

```
int destroy_request (
    struct request * request )
```

@PARAM request: a request @RETURN -1 for fail; 0 for success Destroys and frees a request

References request::args, request::f, request::request, and topologic\_debug.

Referenced by destroy\_graph\_stack().

### 5.14.1.3 procces\_request()

```
int procces_request (
    struct request * request )
```

References edge\_request::a, destroy\_edge\_request::a, destroy\_edge\_id\_request::a, request::args, edge\_request::b, destroy\_edge\_request::b, CREAT\_BI\_EDGE, CREAT\_EDGE, CREAT\_VERTEX, create\_bi\_edge(), create\_edge(), create\_vertex(), DESTROY\_BI\_EDGE, DESTROY\_EDGE, DESTROY\_EDGE\_BY\_ID, DESTROY\_VERTEX, DESTROY\_VERTEX\_BY\_ID, mod\_edge\_vars\_request::edge\_vars, vertex\_request::f, mod\_vertex\_request::f, request::f, edge\_request::f, GENERIC, edge\_request::glbl, vertex\_request::glbl, mod\_vertex\_request::glbl, vertex\_request::graph, destroy\_vertex\_request::graph, destroy\_vertex\_id\_request::graph, destroy\_edge\_id\_request::id, vertex\_request::id, destroy\_vertex\_id\_request::id, MOD\_BI\_EDGE, MOD\_EDGE, MOD\_EDGE\_VARS, MOD\_VERTEX, modify\_bi\_edge(), modify\_edge(), modify\_shared\_edge\_vars(), modify\_vertex(), remove\_bi\_edge(), remove\_edge(), remove\_edge\_id(), remove\_vertex(), remove\_vertex\_id(), request::request, topologic\_debug, mod\_vertex\_request::vertex, mod\_edge\_vars\_request::vertex, and destroy\_vertex\_request::vertex.

Referenced by process\_requests().

### 5.14.1.4 process\_requests()

```
int process_requests (
    struct graph * graph )
```

@PARAM graph: the graph @RETURN 0 if all got processed; -1 if a request failed will set ERRNO to the ENUM Process requests that are queued in the graph

References graph::context, graph::lock, graph::modify, pop(), procces\_request(), graph::remove\_edges, graph::remove\_vertices, SINGLE, and topologic\_debug.

Referenced by run(), and run\_single().

### 5.14.1.5 submit\_request()

```
int submit_request (
    struct graph * graph,
    struct request * request )
```

@PARAM graph: the graph @PARAM request: the request to be processed @RETRUN -1 for fail; 0 for succes;  
Submits a request to be processed after all active nodes complete

References graph::context, DESTROY\_BI\_EDGE, DESTROY\_EDGE, DESTROY\_EDGE\_BY\_ID, DESTROY\_VERTEX, DESTROY\_VERTEX\_BY\_ID, graph::lock, graph::modify, push(), graph::remove\_edges, graph::remove\_vertices, request::request, SINGLE, and topologic\_debug.

## 5.15 src/stack.c File Reference

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

### Functions

- struct stack \* init\_stack ()
- void \* pop (struct stack \*stack)
- void \* get (struct stack \*stack, int index)
- int push (struct stack \*stack, void \*data)
- void destroy\_stack (struct stack \*stack)

### 5.15.1 Function Documentation

#### 5.15.1.1 destroy\_stack()

```
void destroy_stack (
    struct stack * stack )
```

References stack\_node::data, stack::length, stack\_node::next, and stack::root.

Referenced by destroy\_graph\_avl(), destroy\_graph\_stack(), fire(), graph\_init(), print\_edges(), print\_state(), remove\_vertex(), and run\_single().

#### 5.15.1.2 get()

```
void* get (
    struct stack * stack,
    int index )
```

References stack\_node::data, stack::length, stack\_node::next, and stack::root.

### 5.15.1.3 init\_stack()

```
struct stack* init_stack ( )
```

References `stack::length`, and `stack::root`.

Referenced by `destroy_graph_avl()`, `fire()`, `graph_init()`, `print_edges()`, `print_state()`, `remove_vertex()`, and `run_↵single()`.

### 5.15.1.4 pop()

```
void* pop (
    struct stack * stack )
```

References `stack_node::data`, `stack::length`, `stack_node::next`, and `stack::root`.

Referenced by `destroy_graph_avl()`, `destroy_graph_stack()`, `fire()`, `print_edges()`, `print_state()`, `process_requests()`, `remove_vertex()`, `run()`, `run_single()`, and `start_set()`.

### 5.15.1.5 push()

```
int push (
    struct stack * stack,
    void * data )
```

References `stack_node::data`, `stack::length`, `stack_node::next`, and `stack::root`.

Referenced by `inorder_nodes()`, `postorder_nodes()`, `preorder_nodes()`, `stackify_nodes()`, `start_set()`, and `submit_↵request()`.

## 5.16 src/topologic.c File Reference

```
#include "../include/topologic.h"
```

### Functions

- void [sleep\\_ms](#) (int milliseconds)
- int [start\\_set](#) (struct [graph](#) \**graph*, int id[], int num\_vertices)
- int [run\\_single](#) (struct [graph](#) \**graph*, struct [vertex\\_result](#) \*\*init\_vertex\_args)
- int [run](#) (struct [graph](#) \**graph*, struct [vertex\\_result](#) \*\*init\_vertex\_args)
- int [fire](#) (struct [graph](#) \**graph*, struct [vertex](#) \**vertex*, struct [vertex\\_result](#) \*args, enum [STATES](#) color, int iloop)
- void \* [fire\\_pthread](#) (void \*vargp)
- int [switch\\_vertex](#) (struct [graph](#) \**graph*, struct [vertex](#) \**vertex*, struct [vertex\\_result](#) \*args, enum [STATES](#) color, int iloop)
- int [pause\\_graph](#) (struct [graph](#) \**graph*)
- int [resume\\_graph](#) (struct [graph](#) \**graph*)

## 5.16.1 Function Documentation

### 5.16.1.1 fire()

```
int fire (
    struct graph * graph,
    struct vertex * vertex,
    struct vertex_result * args,
    enum STATES color,
    int iloop )
```

@PARAM graph: the graph @PARAM vertex: A vertex to be ran @PARAM args: arguments @PARAM color: the state in which unlocks fire process if STATE is set to PRINT then fire will fail @PARAM iloop: the number of times that vertex fired in succession @RETURNS the result of the vertex fire will wake up the vertex and pass args to the vertex to compute its function and then call switch and clean itself up

References edge::a\_vars, fireable::args, edge::b, edge::b\_vars, BI\_EDGE, edge::bi\_edge\_lock, BLACK, graph::black\_fire, graph::black\_locked, graph::black\_vertex\_count, fireable::color, graph::context, destroy\_stack(), vertex\_result::edge\_argv, vertex::edge\_tree, edge::edge\_type, vertex::f, edge::f, fire(), edge::glbl, vertex::glbl, fireable::iloop, init\_stack(), vertex::is\_active, graph::lock, vertex::lock, graph::max\_loop, graph::max\_state\_changes, NONE, graph::num\_vertices, pop(), preorder(), PTHREAD\_SLEEP\_TIME, RED, graph::red\_fire, graph::red\_locked, graph::red\_vertex\_count, vertex::shared, sleep\_ms(), graph::state, graph::state\_count, SWITCH, SWITCH\_UNSAFE, switch\_vertex(), TERMINATE, topologic\_debug, vertex\_result::vertex\_argv, and shared\_edge::vertex\_data.

Referenced by fire(), and fire\_thread().

### 5.16.1.2 fire\_thread()

```
void* fire_thread (
    void * vargp )
```

Wrapper function for fire, fire\_1 @PARAM vargp: arguments

References fireable::args, fireable::color, fire(), fireable::graph, fireable::iloop, PTHREAD\_SLEEP\_TIME, sleep\_ms(), topologic\_debug, and fireable::vertex.

Referenced by run(), and switch\_vertex().

### 5.16.1.3 pause\_graph()

```
int pause_graph (
    struct graph * graph )
```

@PARAM graph: the graph @RETURN 0 for success -1 if it fails Pauses run

References graph::lock, graph::pause, graph::pause\_cond, and topologic\_debug.



#### 5.16.1.4 resume\_graph()

```
int resume_graph (
    struct graph * graph )
```

@PARAM graph: the graph @RETURN 0 for success -1 if it fails Resumes run

References graph::lock, graph::pause, and topologic\_debug.

#### 5.16.1.5 run()

```
int run (
    struct graph * graph,
    struct vertex_result ** vertex_args )
```

@PARAM graph: the graph @PARAM vertex\_args: array of vertex arguments for f @RETURN 0 if run terminates normally -1 if it fails Attempts to run the graph else aborts.

References ABORT, fireable::args, BLACK, graph::black\_fire, graph::black\_locked, graph::black\_vertex\_count, fireable::color, graph::color\_lock, graph::context, CONTINUE, fire\_pthread(), fireable::graph, fireable::iloop, vertex::is\_active, graph::lock, MAX\_ATTEMPTS, graph::max\_state\_changes, graph::mem\_option, graph::num\_vertices, graph::pause, graph::pause\_cond, pop(), graph::previous\_color, PRINT, graph::print\_flag, print\_graph(), process\_requests(), RED, graph::red\_fire, graph::red\_locked, graph::red\_vertex\_count, run\_single(), SINGLE, graph::start, graph::state, graph::state\_count, TERMINATE, THREAD\_ATTEMPT\_SLEEP, topologic\_debug, fireable::vertex, and WAIT.

#### 5.16.1.6 run\_single()

```
int run_single (
    struct graph * graph,
    struct vertex_result ** init_vertex_args )
```

References edge::a\_vars, edge::b, edge::b\_vars, graph::context, destroy\_stack(), vertex\_result::edge\_argv, vertex::edge\_tree, vertex::f, edge::f, edge::glbl, vertex::glbl, init\_stack(), vertex::is\_active, stack::length, graph::lock, graph::max\_loop, graph::max\_state\_changes, graph::num\_vertices, graph::pause, graph::pause\_cond, pop(), pre-order(), print\_graph(), process\_requests(), vertex::shared, SINGLE, graph::start, graph::state, graph::state\_count, TERMINATE, topologic\_debug, vertex\_result::vertex\_argv, and shared\_edge::vertex\_data.

Referenced by run().

#### 5.16.1.7 sleep\_ms()

```
void sleep_ms (
    int milliseconds )
```

Referenced by fire(), and fire\_pthread().

### 5.16.1.8 start\_set()

```
int start_set (
    struct graph * graph,
    int id[],
    int num_vertices )
```

@PARAM graph: the graph, @PARAM id: the ids of the vertices @PARAM num\_vertices: number of vertices  
@RETURN -1 for fail if any vertex fails; 0 for success Creates multiple contexts that are ran in parallel Handle errors

Given vertex failed, so at this point, free the vertices and leave

References graph::context, find(), vertex::id, pop(), push(), SINGLE, graph::start, topologic\_debug, and graph::vertices.

### 5.16.1.9 switch\_vertex()

```
int switch_vertex (
    struct graph * graph,
    struct vertex * vertex,
    struct vertex_result * args,
    enum STATES color,
    int iloop )
```

@PARAM graph: the graph @PARAM vertex: The vertex in which just fire @PARAM args: The result of the vertex  
Upon call the switch function will compute the edge functions connected to the vertex @PARAM iloop: the number  
of times that vertex fired in succession @RETURNS 0 On success; the vertex connected to the successful edge  
will be fired; -1 on failure

References ABORT, fireable::args, fireable::color, CONTINUE, vertex\_result::edge\_argv, vertex\_result::edge\_size,  
fire\_thread(), fireable::graph, fireable::iloop, MAX\_ATTEMPTS, graph::mem\_option, THREAD\_ATTEMPT\_SLEEP,  
topologic\_debug, fireable::vertex, vertex\_result::vertex\_argv, vertex\_result::vertex\_size, and WAIT.

Referenced by fire().

## 5.17 src/topologic\_json.c File Reference

```
#include "../include/topologic.h"
```

### Functions

- void [print\\_edges](#) (struct [graph](#) \*graph, struct [AVLTree](#) \*edges, const char \*indent, FILE \*out)
- void [print\\_state](#) (struct [graph](#) \*graph, FILE \*out)
- void [print\\_graph](#) (struct [graph](#) \*graph)

### 5.17.1 Function Documentation

### 5.17.1.1 print\_edges()

```
void print_edges (
    struct graph * graph,
    struct AVLTree * edges,
    const char * indent,
    FILE * out )
```

References `edge::a`, `edge::b`, `edge::bi_edge`, `destroy_stack()`, `edge::edge_type`, `edge::f`, `FUNCTIONS`, `edge::glbl`, `GLOBALS`, `edge::id`, `init_stack()`, `stack::length`, `graph::lvl_verbose`, `pop()`, `preorder()`, and `topologic_debug`.

Referenced by `print_state()`.

### 5.17.1.2 print\_graph()

```
void print_graph (
    struct graph * graph )
```

@PARAM graph: the graph Prints the graph with desired output TODO: Print enums

References `graph::context`, `graph::lock`, `graph::lvl_verbose`, `graph::max_loop`, `graph::max_state_changes`, `NO←_VERB`, `graph::num_vertices`, `print_state()`, `SINGLE`, `AVLTree::size`, `graph::snapshot_timestamp`, `START_STOP`, `graph::state_count`, `topologic_debug`, and `graph::vertices`.

Referenced by `run()`, and `run_single()`.

### 5.17.1.3 print\_state()

```
void print_state (
    struct graph * graph,
    FILE * out )
```

References `destroy_stack()`, `vertex::edge_tree`, `EDGES`, `vertex::f`, `FUNCTIONS`, `edge::glbl`, `vertex::glbl`, `GLOBALS`, `vertex::id`, `init_stack()`, `inorder()`, `vertex::is_active`, `stack::length`, `graph::lvl_verbose`, `pop()`, `print_edges()`, `vertex←::shared`, `AVLTree::size`, `topologic_debug`, `VERTICES`, and `graph::vertices`.

Referenced by `print_graph()`.

## 5.18 src/vertex.c File Reference

```
#include "../include/topologic.h"
```

## Functions

- struct `vertex` \* `create_vertex` (struct `graph` \*`graph`, void(\*f)(struct `graph` \*, struct `vertex_result` \*, void \*, void \*), int id, void \*`glbl`)
- int `remove_vertex` (struct `graph` \*`graph`, struct `vertex` \*`vertex`)
- int `remove_vertex_id` (struct `graph` \*`graph`, int id)
- int `modify_vertex` (struct `vertex` \*`vertex`, void(\*f)(struct `graph` \*, struct `vertex_result` \*, void \*, void \*), void \*`glbl`)
- int `modify_shared_edge_vars` (struct `vertex` \*`vertex`, void \*`edge_vars`)

### 5.18.1 Function Documentation

#### 5.18.1.1 `create_vertex()`

```
struct vertex* create_vertex (
    struct graph * graph,
    void(*) (struct graph *, struct vertex_result *, void *, void *) f,
    int id,
    void * glbl )
```

@PARAM graph: the graph @PARAM f: a function @PARAM id: vertex id to be used: MUST BE UNIQUE @PARAM glbl: global variables @RETURN vertex: a vertex to be used in a graph On creation a process will be spawned for the vertex The vertex will compute function f when called NOTE: NULL glbl will mean no global variables. f cannot be NULL.

References `graph::context`, `vertex::context`, `destroy_avl()`, `vertex::edge_tree`, `vertex::f`, `vertex::glbl`, `vertex::id`, `init_avl()`, `insert()`, `vertex::is_active`, `vertex::joining_vertices`, `graph::lock`, `vertex::lock`, `vertex::shared`, `SINGLE`, `topologic_debug`, `shared_edge::vertex_data`, and `graph::vertices`.

Referenced by `procces_request()`.

#### 5.18.1.2 `modify_shared_edge_vars()`

```
int modify_shared_edge_vars (
    struct vertex * vertex,
    void * edge_vars )
```

@PARAM vertex: a vertex @PARAM edge\_vars: shared variables @RETURN 0 for success; -1 for fail Modifies the vertices shared variables with it's edges

References `vertex::context`, `vertex::lock`, `vertex::shared`, `SINGLE`, `topologic_debug`, and `shared_edge::vertex_data`.

Referenced by `procces_request()`.

### 5.18.1.3 modify\_vertex()

```
int modify_vertex (
    struct vertex * vertex,
    void(*) (struct graph *, struct vertex_result *, void *, void *) f,
    void * glbl )
```

@PARAM vertex: a vertex @PARAM f: a function @PARAM glbl: global variables NOTE: NULL f, or glbl will mean no change. @RETURN 0 for success; -1 for fail Modifies the vertices function

References vertex::context, vertex::f, vertex::glbl, vertex::lock, SINGLE, and topologic\_debug.

Referenced by procces\_request().

### 5.18.1.4 remove\_vertex()

```
int remove_vertex (
    struct graph * graph,
    struct vertex * vertex )
```

@PARAM graph: the graph @PARAM vertex: a vertex @RETURN 0 for success; -1 for fail Removes the vertex and all connected edges

References edge::a, edge::b, BI\_EDGE, edge::bi\_edge, edge::bi\_edge\_lock, graph::context, vertex::context, destroy\_stack(), EDGE, vertex::edge\_tree, edge::edge\_type, edge::f, edge::glbl, vertex::glbl, edge::id, vertex::id, init\_stack(), vertex::joining\_vertices, graph::lock, vertex::lock, pop(), remove\_edge\_id(), remove\_ID(), SELF\_EDGE, vertex::shared, SINGLE, stackify(), topologic\_debug, shared\_edge::vertex\_data, and graph::vertices.

Referenced by destroy\_graph\_avl(), procces\_request(), and remove\_vertex\_id().

### 5.18.1.5 remove\_vertex\_id()

```
int remove_vertex_id (
    struct graph * graph,
    int id )
```

@PARAM graph: the graph @PARAM id: the vertex id @RETURN 0 for success; -1 for fail Removes the vertex and all connected edges

References graph::context, find(), graph::lock, remove\_vertex(), SINGLE, topologic\_debug, and graph::vertices.

Referenced by procces\_request().



# Index

- `_GNU_SOURCE`
    - `topologic.h`, 44
- a
  - `destroy_edge_id_request`, 11
  - `destroy_edge_request`, 12
  - `edge`, 14
  - `edge_request`, 16
- `a_vars`
  - `edge`, 14
- `ABORT`
  - `graph.h`, 39
- `args`
  - `fireable`, 17
  - `request`, 25
- `AVL.c`
  - `balance`, 56
  - `create_node`, 56
  - `destroy_avl`, 57
  - `destroy_avl_nodes`, 57
  - `find`, 57
  - `find_node`, 57
  - `init_avl`, 58
  - `inorder`, 58
  - `inorder_nodes`, 58
  - `insert`, 58
  - `insert_node`, 59
  - `left_rotate`, 59
  - `max_height`, 59
  - `minNode`, 59
  - `postorder`, 60
  - `postorder_nodes`, 60
  - `preorder`, 60
  - `preorder_nodes`, 60
  - `remove_ID`, 60
  - `remove_node`, 61
  - `right_rotate`, 61
  - `stackify`, 61
  - `stackify_nodes`, 61
- `AVL.h`
  - `destroy_avl`, 33
  - `find`, 33
  - `init_avl`, 34
  - `inorder`, 34
  - `insert`, 34
  - `postorder`, 34
  - `preorder`, 35
  - `remove_ID`, 35
  - `stackify`, 35
- `AVLNode`, 9
- `data`, 9
- `height`, 9
- `id`, 9
- `left`, 10
- `right`, 10
- `AVLTree`, 10
- `root`, 10
- `size`, 11
- b
  - `destroy_edge_request`, 12
  - `edge`, 14
  - `edge_request`, 16
- `b_vars`
  - `edge`, 14
- `balance`
  - `AVL.c`, 56
- `BI_EDGE`
  - `edge.h`, 37
- `bi_edge`
  - `edge`, 15
- `bi_edge_lock`
  - `edge`, 15
- `BLACK`
  - `graph.h`, 39
- `black_fire`
  - `graph`, 19
- `black_locked`
  - `graph`, 19
- `black_vertex_count`
  - `graph`, 19
- `color`
  - `fireable`, 17
- `color_lock`
  - `graph`, 19
- `CONTEXT`
  - `context.h`, 36
- `context`
  - `graph`, 19
  - `vertex`, 28
- `context.h`
  - `CONTEXT`, 36
  - `NONE`, 37
  - `SINGLE`, 37
  - `SWITCH`, 37
  - `SWITCH_UNSAFE`, 37
- `CONTINUE`
  - `graph.h`, 39
- `CREAT_BI_EDGE`

- request.h, 41
- CREAT\_EDGE
  - request.h, 41
- CREAT\_VERTEX
  - request.h, 41
- CREATE\_BI\_EDGE
  - topologic.h, 44
- create\_bi\_edge
  - edge.c, 62
  - topologic.h, 47
- CREATE\_EDGE
  - topologic.h, 44
- create\_edge
  - edge.c, 62
  - topologic.h, 47
- create\_node
  - AVL.c, 56
- CREATE\_NULL\_BI\_EDGE
  - topologic.h, 44
- CREATE\_REQUEST
  - topologic.h, 44
- create\_request
  - request.c, 66
  - topologic.h, 47
- CREATE\_VERTEX
  - topologic.h, 45
- create\_vertex
  - topologic.h, 48
  - vertex.c, 74
- CREATE\_VERTEX\_GLBL
  - topologic.h, 45
- data
  - AVLNode, 9
  - stack\_node, 27
- destroy\_avl
  - AVL.c, 57
  - AVL.h, 33
- destroy\_avl\_nodes
  - AVL.c, 57
- DESTROY\_BI\_EDGE
  - request.h, 41
- DESTROY\_EDGE
  - request.h, 41
- DESTROY\_EDGE\_BY\_ID
  - request.h, 41
- destroy\_edge\_id\_request, 11
  - a, 11
  - id, 11
- destroy\_edge\_request, 12
  - a, 12
  - b, 12
- destroy\_graph
  - graph.c, 65
  - topologic.h, 48
- destroy\_graph\_avl
  - graph.c, 65
- destroy\_graph\_stack
  - graph.c, 65
- destroy\_request
  - request.c, 66
  - topologic.h, 48
- destroy\_stack
  - stack.c, 68
  - stack.h, 41
- DESTROY\_VERTEX
  - request.h, 41
- DESTROY\_VERTEX\_BY\_ID
  - request.h, 41
- destroy\_vertex\_id\_request, 12
  - graph, 12
  - id, 13
- destroy\_vertex\_request, 13
  - graph, 13
  - vertex, 13
- EDGE
  - edge.h, 37
- edge, 14
  - a, 14
  - a\_vars, 14
  - b, 14
  - b\_vars, 14
  - bi\_edge, 15
  - bi\_edge\_lock, 15
  - edge\_type, 15
  - f, 15
  - glbl, 15
  - id, 15
- edge.c
  - create\_bi\_edge, 62
  - create\_edge, 62
  - modify\_bi\_edge, 63
  - modify\_edge, 63
  - remove\_bi\_edge, 63
  - remove\_edge, 64
  - remove\_edge\_id, 64
- edge.h
  - BI\_EDGE, 37
  - EDGE, 37
  - edge\_type, 37
  - SELF\_EDGE, 37
- edge\_argv
  - vertex\_result, 31
- edge\_data
  - shared\_edge, 26
- edge\_request, 16
  - a, 16
  - b, 16
  - f, 16
  - glbl, 16
- edge\_size
  - vertex\_result, 31
- edge\_tree
  - vertex, 28
- edge\_type
  - edge, 15
  - edge.h, 37



- edge\_vars
  - mod\_edge\_vars\_request, 23
- EDGES
  - graph.h, 40
- f
  - edge, 15
  - edge\_request, 16
  - mod\_vertex\_request, 24
  - request, 25
  - vertex, 28
  - vertex\_request, 30
- find
  - AVL.c, 57
  - AVL.h, 33
- find\_node
  - AVL.c, 57
- fire
  - topologic.c, 70
  - topologic.h, 49
- fire\_pthread
  - topologic.c, 70
  - topologic.h, 49
- fireable, 17
  - args, 17
  - color, 17
  - graph, 17
  - iloop, 17
  - vertex, 18
- FUNCTIONS
  - graph.h, 40
- GENERIC
  - request.h, 41
- get
  - stack.c, 68
  - stack.h, 41
- gbl
  - edge, 15
  - edge\_request, 16
  - mod\_vertex\_request, 24
  - vertex, 28
  - vertex\_request, 30
- GLOBALS
  - graph.h, 40
- graph, 18
  - black\_fire, 19
  - black\_locked, 19
  - black\_vertex\_count, 19
  - color\_lock, 19
  - context, 19
  - destroy\_vertex\_id\_request, 12
  - destroy\_vertex\_request, 13
  - fireable, 17
  - lock, 19
  - lvl\_verbos, 20
  - max\_loop, 20
  - max\_state\_changes, 20
  - mem\_option, 20
  - modify, 20
  - num\_vertices, 20
  - pause, 21
  - pause\_cond, 21
  - previous\_color, 21
  - print\_flag, 21
  - red\_fire, 21
  - red\_locked, 21
  - red\_vertex\_count, 22
  - remove\_edges, 22
  - remove\_vertices, 22
  - snapshot\_timestamp, 22
  - start, 22
  - state, 22
  - state\_count, 23
  - vertex\_request, 30
  - vertices, 23
- graph.c
  - destroy\_graph, 65
  - destroy\_graph\_avl, 65
  - destroy\_graph\_stack, 65
  - graph\_init, 65
- graph.h
  - ABORT, 39
  - BLACK, 39
  - CONTINUE, 39
  - EDGES, 40
  - FUNCTIONS, 40
  - GLOBALS, 40
  - MAX\_ATTEMPTS, 38
  - MEM\_OPTION, 38
  - NO\_SNAP, 39
  - NO\_VERB, 40
  - PRINT, 39
  - RED, 39
  - SNAPSHOT, 39
  - START\_STOP, 39
  - STATES, 39
  - TERMINATE, 39
  - THREAD\_ATTEMPT\_SLEEP, 38
  - VERBOSITY, 39
  - VERTICES, 40
  - WAIT, 39
- GRAPH\_INIT
  - topologic.h, 45
- graph\_init
  - graph.c, 65
  - topologic.h, 49
- height
  - AVLNode, 9
- id
  - AVLNode, 9
  - destroy\_edge\_id\_request, 11
  - destroy\_vertex\_id\_request, 13
  - edge, 15
  - vertex, 29
  - vertex\_request, 30

- illoop
  - fireable, 17
- include/AVL.h, 33
- include/context.h, 35
- include/edge.h, 37
- include/graph.h, 37
- include/header.h, 40
- include/request.h, 40
- include/stack.h, 41
- include/topologic.h, 42
- include/vertex.h, 55
- init\_avl
  - AVL.c, 58
  - AVL.h, 34
- init\_stack
  - stack.c, 68
  - stack.h, 42
- inorder
  - AVL.c, 58
  - AVL.h, 34
- inorder\_nodes
  - AVL.c, 58
- insert
  - AVL.c, 58
  - AVL.h, 34
- insert\_node
  - AVL.c, 59
- is\_active
  - vertex, 29
- joining\_vertices
  - vertex, 29
- left
  - AVLNode, 10
- left\_rotate
  - AVL.c, 59
- length
  - stack, 26
- lock
  - graph, 19
  - vertex, 29
- lvl\_verbose
  - graph, 20
- MAX\_ATTEMPTS
  - graph.h, 38
- max\_height
  - AVL.c, 59
- max\_loop
  - graph, 20
- MAX\_LOOPS
  - topologic.h, 45
- max\_state\_changes
  - graph, 20
- MEM\_OPTION
  - graph.h, 38
- mem\_option
  - graph, 20
- minNode
  - AVL.c, 59
- MOD\_BI\_EDGE
  - request.h, 41
- MOD\_EDGE
  - request.h, 41
- MOD\_EDGE\_VARS
  - request.h, 41
- mod\_edge\_vars\_request, 23
  - edge\_vars, 23
  - vertex, 23
- MOD\_VERTEX
  - request.h, 41
- mod\_vertex\_request, 24
  - f, 24
  - glbl, 24
  - vertex, 24
- modify
  - graph, 20
- MODIFY\_BI\_EDGE
  - topologic.h, 45
- modify\_bi\_edge
  - edge.c, 63
  - topologic.h, 50
- MODIFY\_BI\_EDGE\_GLOBALS
  - topologic.h, 45
- MODIFY\_EDGE
  - topologic.h, 46
- modify\_edge
  - edge.c, 63
  - topologic.h, 50
- MODIFY\_EDGE\_GLOBALS
  - topologic.h, 46
- modify\_shared\_edge\_vars
  - topologic.h, 50
  - vertex.c, 74
- MODIFY\_VERTEX
  - topologic.h, 46
- modify\_vertex
  - topologic.h, 51
  - vertex.c, 74
- MODIFY\_VERTEX\_GLOBALS
  - topologic.h, 46
- next
  - stack\_node, 27
- NO\_SNAP
  - graph.h, 39
- NO\_VERB
  - graph.h, 40
- NONE
  - context.h, 37
- num\_vertices
  - graph, 20
- parse\_json
  - topologic.h, 51
- pause
  - graph, 21

- pause\_cond
  - graph, 21
- pause\_graph
  - topologic.c, 70
  - topologic.h, 51
- pop
  - stack.c, 69
  - stack.h, 42
- postorder
  - AVL.c, 60
  - AVL.h, 34
- postorder\_nodes
  - AVL.c, 60
- preorder
  - AVL.c, 60
  - AVL.h, 35
- preorder\_nodes
  - AVL.c, 60
- previous\_color
  - graph, 21
- PRINT
  - graph.h, 39
- print\_edges
  - topologic\_json.c, 72
- print\_flag
  - graph, 21
- print\_graph
  - topologic.h, 51
  - topologic\_json.c, 73
- print\_state
  - topologic\_json.c, 73
- procces\_request
  - request.c, 67
- process\_requests
  - request.c, 67
  - topologic.h, 52
- PTHREAD\_SLEEP\_TIME
  - topologic.h, 46
- push
  - stack.c, 69
  - stack.h, 42
- README.md, 56
- RED
  - graph.h, 39
- red\_fire
  - graph, 21
- red\_locked
  - graph, 21
- red\_vertex\_count
  - graph, 22
- remove\_bi\_edge
  - edge.c, 63
  - topologic.h, 52
- remove\_edge
  - edge.c, 64
  - topologic.h, 52
- remove\_edge\_id
  - edge.c, 64
- topologic.h, 52
- remove\_edges
  - graph, 22
- remove\_ID
  - AVL.c, 60
  - AVL.h, 35
- remove\_node
  - AVL.c, 61
- remove\_vertex
  - topologic.h, 53
  - vertex.c, 75
- remove\_vertex\_id
  - topologic.h, 53
  - vertex.c, 75
- remove\_vertices
  - graph, 22
- request, 25
  - args, 25
  - f, 25
  - request, 25
- request.c
  - create\_request, 66
  - destroy\_request, 66
  - procces\_request, 67
  - process\_requests, 67
  - submit\_request, 67
- request.h
  - CREAT\_BI\_EDGE, 41
  - CREAT\_EDGE, 41
  - CREAT\_VERTEX, 41
  - DESTROY\_BI\_EDGE, 41
  - DESTROY\_EDGE, 41
  - DESTROY\_EDGE\_BY\_ID, 41
  - DESTROY\_VERTEX, 41
  - DESTROY\_VERTEX\_BY\_ID, 41
  - GENERIC, 41
  - MOD\_BI\_EDGE, 41
  - MOD\_EDGE, 41
  - MOD\_EDGE\_VARS, 41
  - MOD\_VERTEX, 41
  - REQUESTS, 40
- REQUESTS
  - request.h, 40
- resume\_graph
  - topologic.c, 70
  - topologic.h, 53
- right
  - AVLNode, 10
- right\_rotate
  - AVL.c, 61
- root
  - AVLTree, 10
  - stack, 27
- run
  - topologic.c, 71
  - topologic.h, 54
- run\_single
  - topologic.c, 71

- SELF\_EDGE
  - edge.h, 37
- shared
  - vertex, 29
- shared\_edge, 26
  - edge\_data, 26
  - vertex\_data, 26
- SINGLE
  - context.h, 37
- size
  - AVLTree, 11
- sleep\_ms
  - topologic.c, 71
- SNAPSHOT
  - graph.h, 39
- snapshot\_timestamp
  - graph, 22
- src/AVL.c, 56
- src/edge.c, 62
- src/graph.c, 64
- src/request.c, 66
- src/stack.c, 68
- src/topologic.c, 69
- src/topologic\_json.c, 72
- src/vertex.c, 73
- stack, 26
  - length, 26
  - root, 27
- stack.c
  - destroy\_stack, 68
  - get, 68
  - init\_stack, 68
  - pop, 69
  - push, 69
- stack.h
  - destroy\_stack, 41
  - get, 41
  - init\_stack, 42
  - pop, 42
  - push, 42
- stack\_node, 27
  - data, 27
  - next, 27
- stackify
  - AVL.c, 61
  - AVL.h, 35
- stackify\_nodes
  - AVL.c, 61
- start
  - graph, 22
- start\_set
  - topologic.c, 71
  - topologic.h, 54
- START\_STOP
  - graph.h, 39
- state
  - graph, 22
- state\_count
  - graph, 23
- STATES
  - graph.h, 39
- submit\_request
  - request.c, 67
  - topologic.h, 54
- SWITCH
  - context.h, 37
- SWITCH\_UNSAFE
  - context.h, 37
- switch\_vertex
  - topologic.c, 72
  - topologic.h, 55
- TERMINATE
  - graph.h, 39
- THREAD\_ATTEMPT\_SLEEP
  - graph.h, 38
- topologic.c
  - fire, 70
  - fire\_pthread, 70
  - pause\_graph, 70
  - resume\_graph, 70
  - run, 71
  - run\_single, 71
  - sleep\_ms, 71
  - start\_set, 71
  - switch\_vertex, 72
- topologic.h
  - \_GNU\_SOURCE, 44
  - CREATE\_BI\_EDGE, 44
  - create\_bi\_edge, 47
  - CREATE\_EDGE, 44
  - create\_edge, 47
  - CREATE\_NULL\_BI\_EDGE, 44
  - CREATE\_REQUEST, 44
  - create\_request, 47
  - CREATE\_VERTEX, 45
  - create\_vertex, 48
  - CREATE\_VERTEX\_GLBL, 45
  - destroy\_graph, 48
  - destroy\_request, 48
  - fire, 49
  - fire\_pthread, 49
  - GRAPH\_INIT, 45
  - graph\_init, 49
  - MAX\_LOOPS, 45
  - MODIFY\_BI\_EDGE, 45
  - modify\_bi\_edge, 50
  - MODIFY\_BI\_EDGE\_GLOBALS, 45
  - MODIFY\_EDGE, 46
  - modify\_edge, 50
  - MODIFY\_EDGE\_GLOBALS, 46
  - modify\_shared\_edge\_vars, 50
  - MODIFY\_VERTEX, 46
  - modify\_vertex, 51
  - MODIFY\_VERTEX\_GLOBALS, 46
  - parse\_json, 51
  - pause\_graph, 51

- print\_graph, 51
- process\_requests, 52
- PTHREAD\_SLEEP\_TIME, 46
- remove\_bi\_edge, 52
- remove\_edge, 52
- remove\_edge\_id, 52
- remove\_vertex, 53
- remove\_vertex\_id, 53
- resume\_graph, 53
- run, 54
- start\_set, 54
- submit\_request, 54
- switch\_vertex, 55
- TOPOLOGIC\_DEBUG, 46
- topologic\_debug, 47
- TOPOLOGIC\_DEBUG
  - topologic.h, 46
- topologic\_debug
  - topologic.h, 47
- topologic\_json.c
  - print\_edges, 72
  - print\_graph, 73
  - print\_state, 73
- VERBOSITY
  - graph.h, 39
- vertex, 28
  - context, 28
  - destroy\_vertex\_request, 13
  - edge\_tree, 28
  - f, 28
  - fireable, 18
  - glbl, 28
  - id, 29
  - is\_active, 29
  - joining\_vertices, 29
  - lock, 29
  - mod\_edge\_vars\_request, 23
  - mod\_vertex\_request, 24
  - shared, 29
- vertex.c
  - create\_vertex, 74
  - modify\_shared\_edge\_vars, 74
  - modify\_vertex, 74
  - remove\_vertex, 75
  - remove\_vertex\_id, 75
- vertex\_argv
  - vertex\_result, 31
- vertex\_data
  - shared\_edge, 26
- vertex\_request, 30
  - f, 30
  - glbl, 30
  - graph, 30
  - id, 30
- vertex\_result, 31
  - edge\_argv, 31
  - edge\_size, 31
  - vertex\_argv, 31
  - vertex\_size, 31
- vertex\_size
- vertex\_result, 31
- VERTICES
  - graph.h, 40
- vertices
  - graph, 23
- WAIT
  - graph.h, 39