

Graph

beta

Generated by Doxygen 1.10.0

1 Data Structure Index	1
1.1 Data Structures	1
2 File Index	3
2.1 File List	3
3 Data Structure Documentation	5
3.1 edge Struct Reference	5
3.1.1 Detailed Description	5
3.1.2 Field Documentation	5
3.1.2.1 end_vertex	5
3.1.2.2 length	5
3.1.2.3 start_vertex	6
3.2 graph Struct Reference	6
3.2.1 Detailed Description	6
3.2.2 Field Documentation	7
3.2.2.1 edges	7
3.2.2.2 edges_amount	7
3.2.2.3 vertices	7
3.2.2.4 vertices_amount	7
4 File Documentation	9
4.1 D:/files from internet/important/learning/github/c-modules/Graph/code/inc/graph.h File Reference . . .	9
4.1.1 Macro Definition Documentation	11
4.1.1.1 _GRAPH_EMPTY__	11
4.1.1.2 _GRAPH_EXIST__	11
4.1.1.3 _GRAPH_FORBIDDEN_SEPARATORS__	12
4.1.1.4 _GRAPH_INCORRECT_ARG__	12
4.1.1.5 _GRAPH_MEM__	12
4.1.1.6 _GRAPH_NOT_FOUND__	12
4.1.1.7 _GRAPH_OK__	12
4.1.1.8 _GRAPH_OS_ERROR__	12
4.1.1.9 _STRING__	12
4.1.2 Typedef Documentation	13
4.1.2.1 graph_error_t	13
4.1.3 Function Documentation	13
4.1.3.1 graph_add_edge()	13
4.1.3.2 graph_add_vertex()	14
4.1.3.3 graph_adjacency_list_fill()	14
4.1.3.4 graph_adjacency_list_size()	15
4.1.3.5 graph_delete_edge()	15
4.1.3.6 graph_delete_vertex()	16
4.1.3.7 graph_dfs()	17

4.1.3.8 graph_free()	17
4.1.3.9 graph_has_edge()	17
4.1.3.10 graph_has_vertex()	18
4.1.3.11 graph_initialize()	19
4.1.3.12 graph_is_empty()	20
4.1.3.13 graph_show()	20
4.1.3.14 graph_to_dot()	21
4.2 graph.h	22
4.3 D:/files from internet/important/learning/github/c-modules/Graph/code/src/graph.c File Reference	23
4.3.1 Function Documentation	24
4.3.1.1 graph_add_edge()	24
4.3.1.2 graph_add_vertex()	25
4.3.1.3 graph_adjacency_list_fill()	26
4.3.1.4 graph_adjacency_list_size()	26
4.3.1.5 graph_delete_edge()	27
4.3.1.6 graph_delete_vertex()	28
4.3.1.7 graph_dfs()	28
4.3.1.8 graph_free()	28
4.3.1.9 graph_has_edge()	29
4.3.1.10 graph_has_vertex()	30
4.3.1.11 graph_initialize()	30
4.3.1.12 graph_is_empty()	31
4.3.1.13 graph_show()	31
4.3.1.14 graph_to_dot()	32
Index	35

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

edge	Edge of graph	5
graph	Graph	6

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

D:/files from internet/important/learning/github/c-modules/Graph/code/inc/ graph.h	9
D:/files from internet/important/learning/github/c-modules/Graph/code/src/ graph.c	23

Chapter 3

Data Structure Documentation

3.1 edge Struct Reference

Edge of graph.

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

Data Fields

- char **start_vertex** [**_STRING__**+1]
- char **end_vertex** [**_STRING__**+1]
- size_t **length**

3.1.1 Detailed Description

Edge of graph.

Parameters

<i>start_vertex</i>	Name of start vertex
<i>end_vertex</i>	Name of end vertex
<i>length</i>	Length of edge

3.1.2 Field Documentation

3.1.2.1 end_vertex

```
char end_vertex[ _STRING__+1]
```

3.1.2.2 length

```
size_t length
```

3.1.2.3 start_vertex

```
char start_vertex[ _STRING__+1]
```

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

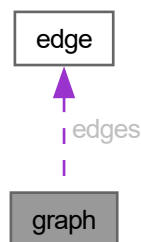
- D:/files from internet/important/learning/github/c-modules/Graph/code/inc/ **graph.h**

3.2 graph Struct Reference

Graph.

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

Collaboration diagram for graph:



Data Fields

- char ** **vertices**
- size_t **vertices_amount**
- struct **edge** * **edges**
- size_t **edges_amount**

3.2.1 Detailed Description

Graph.

Parameters

<i>vertices</i>	Dynamic array of vertices names
<i>vertices_amount</i>	Length of vertices array
<i>edges</i>	Dynamic array of edges
<i>edges_amount</i>	Length of edges array

3.2.2 Field Documentation

3.2.2.1 edges

```
struct edge* edges
```

3.2.2.2 edges_amount

```
size_t edges_amount
```

3.2.2.3 vertices

```
char** vertices
```

3.2.2.4 vertices_amount

```
size_t vertices_amount
```

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

- D:/files from internet/important/learning/github/c-modules/Graph/code/inc/ **graph.h**

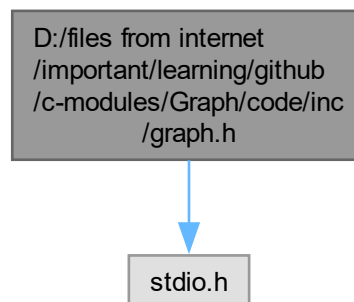
Chapter 4

File Documentation

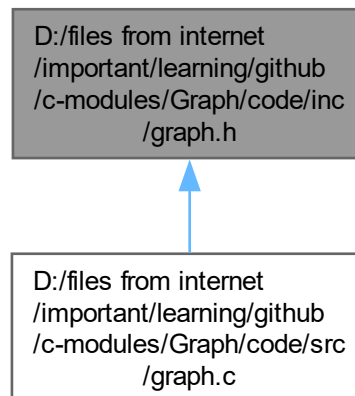
4.1 D:/files from internet/important/learning/github/c-modules/ Graph/code/inc/graph.h File Reference

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

Include dependency graph for graph.h:



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



Data Structures

- struct **edge**
Edge of graph.
- struct **graph**
Graph.

Macros

- #define **_STRING__** 256
- #define **_GRAPH_FORBIDDEN_SEPARATORS__** "\"'#{%()}><{}-\\|:;,"
- #define **_GRAPH_OK__** 0
Positive return code.
- #define **_GRAPH_MEM__** -1
Memory shortage error.
- #define **_GRAPH_INCORRECT_ARG__** -2
Incorrect arguments in function.
- #define **_GRAPH_EMPTY__** -3
Graph is empty.
- #define **_GRAPH_NOT_FOUND__** -4
Object not in graph.
- #define **_GRAPH_EXIST__** -5
Graph already has object.
- #define **_GRAPH_OS_ERROR__** -6
Operating system error.

Typedefs

- typedef int **graph_error_t**
Data type for errors that occur during the operation of functions.

Functions

- void **graph_initialize** (struct **graph** * **graph**)
Initialization of graph by zero.
- int **graph_is_empty** (const struct **graph** * **graph**)
Checking for graph emptiness.
- int **graph_has_vertex** (const struct **graph** * **graph**, const char *vertex)
Checking for the presence of a vertex in the graph.
- int **graph_has_edge** (const struct **graph** * **graph**, const char *start_vertex, const char *end_vertex)
Checking for the presence of a edge in the graph.
- **graph_error_t** **graph_add_vertex** (struct **graph** * **graph**, const char *vertex)
Adding a vertex to a graph.
- **graph_error_t** **graph_delete_vertex** (struct **graph** * **graph**, const char *vertex)
Deleting vertex from graph.
- **graph_error_t** **graph_add_edge** (struct **graph** * **graph**, const char *start_vertex, const char *end_vertex, size_t edge_length)
Adding an edge to a graph.
- **graph_error_t** **graph_delete_edge** (struct **graph** * **graph**, const char *start_vertex, const char *end_vertex)
Deleting edge from graph.
- **graph_error_t** **graph_show** (const struct **graph** * **graph**)
Draw graph using Graphviz and show it.
- **graph_error_t** **graph_to_dot** (const struct **graph** * **graph**, const char *folder, const char *filename)
Creating a dot file by graph.
- size_t **graph_adjacency_list_size** (const struct **graph** * **graph**, const char *vertex)
Counting the number of adjacent vertices (the size of the adjacency list)
- **graph_error_t** **graph_adjacency_list_fill** (const struct **graph** * **graph**, const char *vertex, int *adjacency_list)
Filling in the adjacency list.
- void **graph_dfs** (struct **graph** * **graph**, void(*vertex_processing)(char *vertex_name))
Graph traversal using a depth-first search algorithm.
- void **graph_free** (struct **graph** * **graph**)
Free graph.

4.1.1 Macro Definition Documentation

4.1.1.1 _GRAPH_EMPTY__

```
#define _GRAPH_EMPTY__ -3
```

Graph is empty.

4.1.1.2 _GRAPH_EXIST__

```
#define _GRAPH_EXIST__ -5
```

Graph already has object.

4.1.1.3 `_GRAPH_FORBIDDEN_SEPARATORS__`

```
#define _GRAPH_FORBIDDEN_SEPARATORS__ "\"'()%><{}-/\|:;,"
```

Forbidden characters for vertex name

4.1.1.4 `_GRAPH_INCORRECT_ARG__`

```
#define _GRAPH_INCORRECT_ARG__ -2
```

Incorrect arguments in function.

4.1.1.5 `_GRAPH_MEM__`

```
#define _GRAPH_MEM__ -1
```

Memory shortage error.

4.1.1.6 `_GRAPH_NOT_FOUND__`

```
#define _GRAPH_NOT_FOUND__ -4
```

Object not in graph.

4.1.1.7 `_GRAPH_OK__`

```
#define _GRAPH_OK__ 0
```

Positive return code.

4.1.1.8 `_GRAPH_OS_ERROR__`

```
#define _GRAPH_OS_ERROR__ -6
```

Operating system error.

4.1.1.9 `_STRING__`

```
#define _STRING__ 256
```

Max length of string

4.1.2 Typedef Documentation

4.1.2.1 graph_error_t

```
typedef int graph_error_t
```

Data type for errors that occur during the operation of functions.

4.1.3 Function Documentation

4.1.3.1 graph_add_edge()

```
graph_error_t graph_add_edge (
    struct graph * graph,
    const char * start_vertex,
    const char * end_vertex,
    size_t edge_length )
```

Adding an edge to a graph.

Parameters

in	<i>graph</i>	Graph descriptor
in	<i>start_vertex</i>	Start vertex name
in	<i>end_vertex</i>	End vertex name

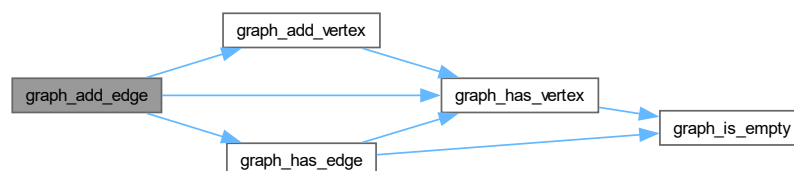
Returns

`__GRAPH_OK__`, `__GRAPH_MEM__`, `__GRAPH_INCORRECT_ARG__`, `__GRAPH_EXIST__`

Note

- You cannot add a copy of an existing edge
- When adding an edge consisting of new vertices, new vertices will be added to the graph

Here is the call graph for this function:



4.1.3.2 graph_add_vertex()

```
graph_error_t graph_add_vertex (
    struct graph * graph,
    const char * vertex )
```

Adding a vertex to a graph.

Parameters

in	<i>graph</i>	Graph descriptor
in	<i>vertex</i>	Vertex name

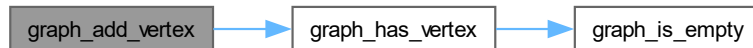
Returns

`__GRAPH_OK__`, `__GRAPH_MEM__`, `__GRAPH_INCORRECT_ARG__`, `__GRAPH_EXIST__`

Note

- You cannot add a copy of an existing vertex
- You cannot add a vertex with a name of zero length
- You cannot add a vertex with a name containing special characters - `# % () > < { } - / \ | : ; ,` and quotes

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.3 graph_adjacency_list_fill()

```
graph_error_t graph_adjacency_list_fill (
    const struct graph * graph,
    const char * vertex,
    int * adjacency_list )
```

Filling in the adjacency list.

Parameters

in	<i>graph</i>	Graph descriptor
in	<i>vertex</i>	Vertex name
in	<i>size</i>	Adjacency list size
out	<i>adjacency_list</i>	Adjacency list descriptor

Returns

`_GRAPH_OK__`, `_GRAPH_INCORRECT_ARG__`

4.1.3.4 `graph_adjacency_list_size()`

```
size_t graph_adjacency_list_size (
    const struct graph * graph,
    const char * vertex )
```

Counting the number of adjacent vertices (the size of the adjacency list)

Parameters

in	<i>graph</i>	Graph descriptor
in	<i>vertex</i>	Vertex name

Returns

The number of adjacent vertices

Note

- If the arguments is incorrect, the function returns 0

Here is the call graph for this function:

4.1.3.5 `graph_delete_edge()`

```
graph_error_t graph_delete_edge (
    struct graph * graph,
    const char * start_vertex,
    const char * end_vertex )
```

Deleting edge from graph.

Parameters

in	<i>graph</i>	Graph descriptor
in	<i>start_vertex</i>	Start vertex name
in	<i>end_vertex</i>	End vertex name

Returns

`__GRAPH_OK__`, `__GRAPH_INCORRECT_ARG__`, `__GRAPH_EMPTY__`, `__GRAPH_NOT_FOUND__`

Here is the call graph for this function:



Here is the caller graph for this function:

**4.1.3.6 graph_delete_vertex()**

```

graph_error_t graph_delete_vertex (
    struct graph * graph,
    const char * vertex )
  
```

Deleting vertex from graph.

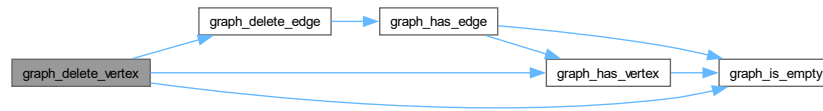
Parameters

in	<i>graph</i>	Graph descriptor
in	<i>vertex</i>	Vertex name

Returns

`_GRAPH_OK__`, `_GRAPH_INCORRECT_ARG__`, `_GRAPH_EMPTY__`, `_GRAPH_NOT_FOUND__`

Here is the call graph for this function:

**4.1.3.7 graph_dfs()**

```
void graph_dfs (
    struct graph * graph,
    void(*) (char *vertex_name) vertex_processing )
```

Graph traversal using a depth-first search algorithm.

Parameters

in	<i>graph</i>	Graph descriptor
in	<i>vertex_processing</i>	Vertex processing function

Note

- If the input arguments are incorrect, the function will not work

4.1.3.8 graph_free()

```
void graph_free (
    struct graph * graph )
```

Free graph.

Parameters

in	<i>graph</i>	Graph descriptor
----	--------------	------------------

4.1.3.9 graph_has_edge()

```
int graph_has_edge (
    const struct graph * graph,
```

```
const char * start_vertex,
const char * end_vertex )
```

Checking for the presence of a edge in the graph.

Parameters

in	<i>graph</i>	Graph descriptor
in	<i>start_vertex</i>	Start vertex name
in	<i>end_vertex</i>	End vertex name

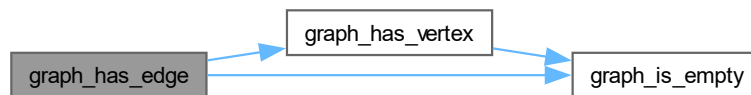
Returns

1 - True / 0 - False

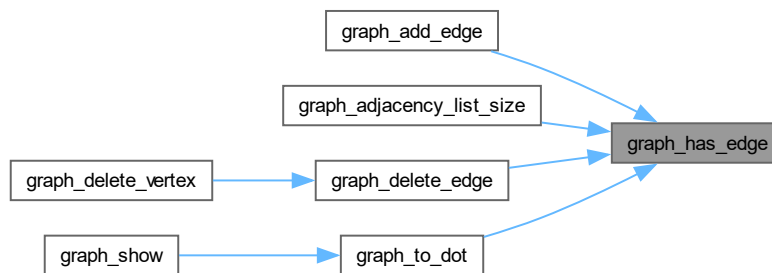
Note

- If incorrect arguments are passed, the function returns 0 (False)

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.10 graph_has_vertex()

```
int graph_has_vertex (
    const struct graph * graph,
    const char * vertex )
```

Checking for the presence of a vertex in the graph.

Parameters

in	<i>graph</i>	Graph descriptor
in	<i>vertex</i>	Vertex name

Returns

1 - True / 0 - False

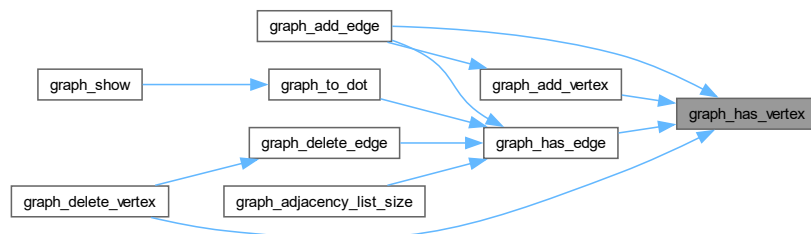
Note

- If incorrect arguments are passed, the function returns 0 (False)

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.11 graph_initialize()

```
void graph_initialize (
    struct graph * graph )
```

Initialization of graph by zero.

Parameters

in	<i>graph</i>	Graph descriptor
----	--------------	------------------

Note

- If the graph descriptor is NULL, the function will not cause a segmentation error

4.1.3.12 graph_is_empty()

```
int graph_is_empty (
    const struct graph * graph )
```

Checking for graph emtiness.

Parameters

in	<i>graph</i>	Graph descriptor
----	--------------	------------------

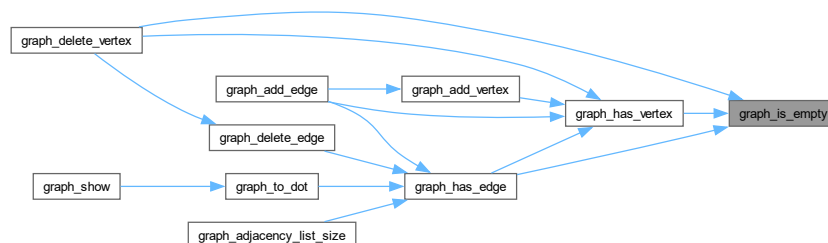
Returns

1 - True / 0 - False

Note

- If incorrect arguments are passed, the function returns 1 (True)

Here is the caller graph for this function:

**4.1.3.13 graph_show()**

```
graph_error_t graph_show (
    const struct graph * graph )
```

Draw graph using Graphviz and show it.

Parameters

in	<i>graph</i>	Graph descriptor
----	--------------	------------------

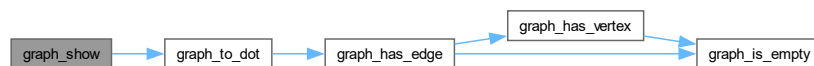
Returns

`__GRAPH_OK__`, `__GRAPH_MEM__`, `__GRAPH_INCORRECT_ARGS__`, `__GRAPH_OS_ERROR__`

Note

- Linux: the graph is demonstrated using `eog`
- Windows: the graph is demonstrated using `mspaint`
- The function creates a separate folder for temporary files and deletes it at the end of the work

Here is the call graph for this function:



4.1.3.14 graph_to_dot()

```

graph_error_t graph_to_dot (
    const struct graph * graph,
    const char * folder,
    const char * filename )
  
```

Creating a dot file by graph.

Parameters

in	<i>graph</i>	Graph descriptor
in	<i>folder</i>	Folder name
in	<i>filename</i>	File name

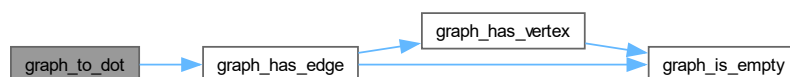
Returns

`__GRAPH_OK__`, `__GRAPH_INCORRECT_ARG__`, `__GRAPH_MEM__`, `__GRAPH_OS_ERROR__`

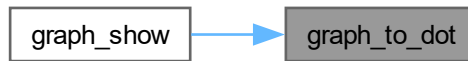
Note

- The pointer to the `folder` string can take the `NULL` value. In this case, the folder will not be created

Here is the call graph for this function:



Here is the caller graph for this function:



4.2 graph.h

Go to the documentation of this file.

```

00001 #ifndef GRAPH_H__
00002 #define GRAPH_H__
00003
00004 #include <stdio.h>
00005
00006 // Macro
00007
00011 #define _STRING__ 256
00012
00016 #define _GRAPH_FORBIDDEN_SEPARATORS__ "\"'()%><{}-/\|:;,"
00017
00021 #define _GRAPH_OK__ 0
00022
00026 #define _GRAPH_MEM__ -1
00027
00031 #define _GRAPH_INCORRECT_ARG__ -2
00032
00036 #define _GRAPH_EMPTY__ -3
00037
00041 #define _GRAPH_NOT_FOUND__ -4
00042
00046 #define _GRAPH_EXIST__ -5
00047
00051 #define _GRAPH_OS_ERROR__ -6
00052
00053 // Structs and functions
00054
00062 struct edge
00063 {
00064     char start_vertex[_STRING__ + 1];
00065     char end_vertex[_STRING__ + 1];
00066     size_t length;
00067 };
00068
00077 struct graph
00078 {
00079     char **vertices;
00080     size_t vertices_amount;
00081     struct edge *edges;
00082     size_t edges_amount;
00083 };
00084
00088 typedef int graph_error_t;
00089
00097 void graph_initialize(struct graph *graph);
00098
00108 int graph_is_empty(const struct graph *graph);
00109
00120 int graph_has_vertex(const struct graph *graph, const char *vertex);
00121
00133 int graph_has_edge(const struct graph *graph, const char *start_vertex, const char *end_vertex);
00134
00147 graph_error_t graph_add_vertex(struct graph *graph, const char *vertex);
00148
00157 graph_error_t graph_delete_vertex(struct graph *graph, const char *vertex);
00158
00171 graph_error_t graph_add_edge(struct graph *graph, const char *start_vertex, const char *end_vertex,
    size_t edge_length);
00172
00182 graph_error_t graph_delete_edge(struct graph *graph, const char *start_vertex, const char
    *end_vertex);
  
```

```

00183
00195 graph_error_t graph_show(const struct graph *graph);
00196
00208 graph_error_t graph_to_dot(const struct graph *graph, const char *folder, const char *filename);
00209
00220 size_t graph_adjacency_list_size(const struct graph *graph, const char *vertex);
00221
00233 graph_error_t graph_adjacency_list_fill(const struct graph *graph, const char *vertex, int
    *adjacency_list);
00234
00243 void graph_dfs(struct graph *graph, void (*vertex_processing)(char *vertex_name));
00244
00250 void graph_free(struct graph *graph);
00251
00252 #endif // GRAPH_H__

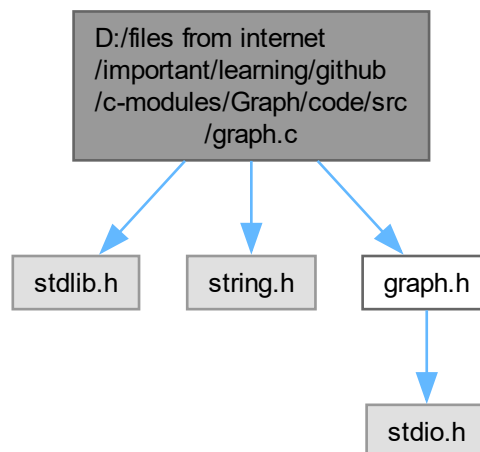
```

4.3 D:/files from internet/important/learning/github/c-modules/Graph/code/src/graph.c File Reference

```

#include <stdlib.h>
#include <string.h>
#include "graph.h"
Include dependency graph for graph.c:

```



Functions

- void **graph_initialize** (struct **graph** * **graph**)
Initialization of graph by zero.
- int **graph_is_empty** (const struct **graph** * **graph**)
Checking for graph emptiness.
- int **graph_has_vertex** (const struct **graph** * **graph**, const char *vertex)
Checking for the presence of a vertex in the graph.
- int **graph_has_edge** (const struct **graph** * **graph**, const char *start_vertex, const char *end_vertex)
Checking for the presence of a edge in the graph.
- **graph_error_t** **graph_add_vertex** (struct **graph** * **graph**, const char *vertex)

- Adding a vertex to a graph.*

 - **graph_error_t graph_delete_vertex** (struct **graph** * **graph**, const char *vertex)

Deleting vertex from graph.
- **graph_error_t graph_add_edge** (struct **graph** * **graph**, const char *start_vertex, const char *end_vertex, size_t edge_length)

Adding an edge to a graph.
- **graph_error_t graph_delete_edge** (struct **graph** * **graph**, const char *start_vertex, const char *end_vertex)

Deleting edge from graph.
- **graph_error_t graph_to_dot** (const struct **graph** * **graph**, const char *folder, const char *filename)

Creating a dot file by graph.
- **graph_error_t graph_show** (const struct **graph** * **graph**)

Draw graph using Graphviz and show it.
- size_t **graph_adjacency_list_size** (const struct **graph** * **graph**, const char *vertex)

Counting the number of adjacent vertices (the size of the adjacency list)
- **graph_error_t graph_adjacency_list_fill** (const struct **graph** * **graph**, const char *vertex, int *adjacency_list)

Filling in the adjacency list.
- void **graph_dfs** (struct **graph** * **graph**, void(*vertex_processing)(char *vertex_name))

Graph traversal using a depth-first search algorithm.
- void **graph_free** (struct **graph** * **graph**)

Free graph.

4.3.1 Function Documentation

4.3.1.1 graph_add_edge()

```
graph_error_t graph_add_edge (
    struct graph * graph,
    const char * start_vertex,
    const char * end_vertex,
    size_t edge_length )
```

Adding an edge to a graph.

Parameters

in	<i>graph</i>	Graph descriptor
in	<i>start_vertex</i>	Start vertex name
in	<i>end_vertex</i>	End vertex name

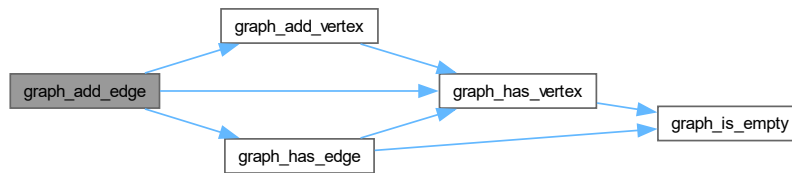
Returns

`_GRAPH_OK__`, `_GRAPH_MEM__`, `_GRAPH_INCORRECT_ARG__`, `_GRAPH_EXIST__`

Note

- You cannot add a copy of an existing edge
- When adding an edge consisting of new vertices, new vertices will be added to the graph

Here is the call graph for this function:



4.3.1.2 graph_add_vertex()

```

graph_error_t graph_add_vertex (
    struct graph * graph,
    const char * vertex )

```

Adding a vertex to a graph.

Parameters

in	<i>graph</i>	Graph descriptor
in	<i>vertex</i>	Vertex name

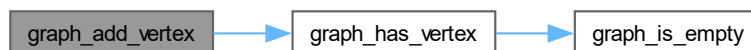
Returns

`_GRAPH_OK__`, `_GRAPH_MEM__`, `_GRAPH_INCORRECT_ARG__`, `_GRAPH_EXIST__`

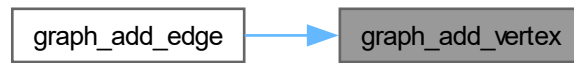
Note

- You cannot add a copy of an existing vertex
- You cannot add a vertex with a name of zero length
- You cannot add a vertex with a name containing special characters - `# % () > < { } - / \ | : ; ,` and quotes

Here is the call graph for this function:



Here is the caller graph for this function:



4.3.1.3 graph_adjacency_list_fill()

```

graph_error_t graph_adjacency_list_fill (
    const struct graph * graph,
    const char * vertex,
    int * adjacency_list )
  
```

Filling in the adjacency list.

Parameters

in	<i>graph</i>	Graph descriptor
in	<i>vertex</i>	Vertex name
in	<i>size</i>	Adjacency list size
out	<i>adjacency_list</i>	Adjacency list descriptor

Returns

`_GRAPH_OK__`, `_GRAPH_INCORRECT_ARG__`

4.3.1.4 graph_adjacency_list_size()

```

size_t graph_adjacency_list_size (
    const struct graph * graph,
    const char * vertex )
  
```

Counting the number of adjacent vertices (the size of the adjacency list)

Parameters

in	<i>graph</i>	Graph descriptor
in	<i>vertex</i>	Vertex name

Returns

The number of adjacent vertices

Note

- If the arguments is incorrect, the function returns 0

Here is the call graph for this function:

**4.3.1.5 graph_delete_edge()**

```

graph_error_t graph_delete_edge (
    struct graph * graph,
    const char * start_vertex,
    const char * end_vertex )
  
```

Deleting edge from graph.

Parameters

in	<i>graph</i>	Graph descriptor
in	<i>start_vertex</i>	Start vertex name
in	<i>end_vertex</i>	End vertex name

Returns

`_GRAPH_OK__`, `_GRAPH_INCORRECT_ARG__`, `_GRAPH_EMPTY__`, `_GRAPH_NOT_FOUND__`

Here is the call graph for this function:



Here is the caller graph for this function:



4.3.1.6 graph_delete_vertex()

```
graph_error_t graph_delete_vertex (
    struct graph * graph,
    const char * vertex )
```

Deleting vertex from graph.

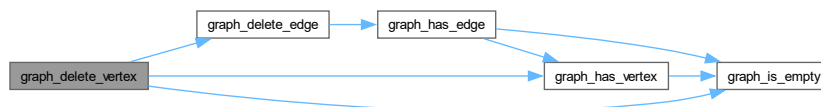
Parameters

in	<i>graph</i>	Graph descriptor
in	<i>vertex</i>	Vertex name

Returns

`_GRAPH_OK__`, `_GRAPH_INCORRECT_ARG__`, `_GRAPH_EMPTY__`, `_GRAPH_NOT_FOUND__`

Here is the call graph for this function:



4.3.1.7 graph_dfs()

```
void graph_dfs (
    struct graph * graph,
    void(*) (char *vertex_name) vertex_processing )
```

Graph traversal using a depth-first search algorithm.

Parameters

in	<i>graph</i>	Graph descriptor
in	<i>vertex_processing</i>	Vertex processing function

Note

- If the input arguments are incorrect, the function will not work

4.3.1.8 graph_free()

```
void graph_free (
    struct graph * graph )
```

Free graph.

Parameters

in	<i>graph</i>	Graph descriptor
----	--------------	------------------

4.3.1.9 graph_has_edge()

```
int graph_has_edge (
    const struct graph * graph,
    const char * start_vertex,
    const char * end_vertex )
```

Checking for the presence of a edge in the graph.

Parameters

in	<i>graph</i>	Graph descriptor
in	<i>start_vertex</i>	Start vertex name
in	<i>end_vertex</i>	End vertex name

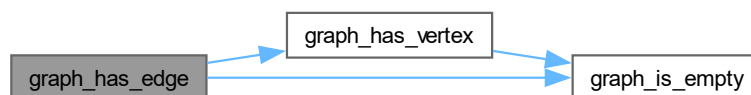
Returns

1 - True / 0 - False

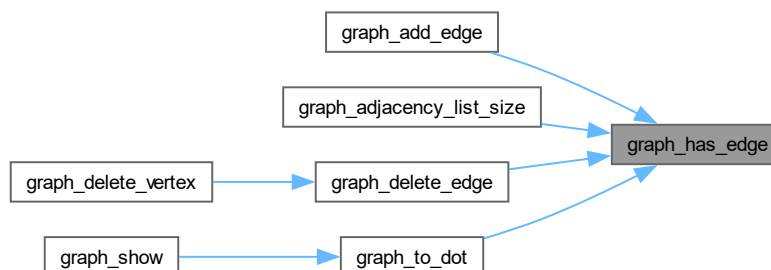
Note

- If incorrect arguments are passed, the function returns 0 (False)

Here is the call graph for this function:



Here is the caller graph for this function:



4.3.1.10 graph_has_vertex()

```
int graph_has_vertex (
    const struct graph * graph,
    const char * vertex )
```

Checking for the presence of a vertex in the graph.

Parameters

in	<i>graph</i>	Graph descriptor
in	<i>vertex</i>	Vertex name

Returns

1 - True / 0 - False

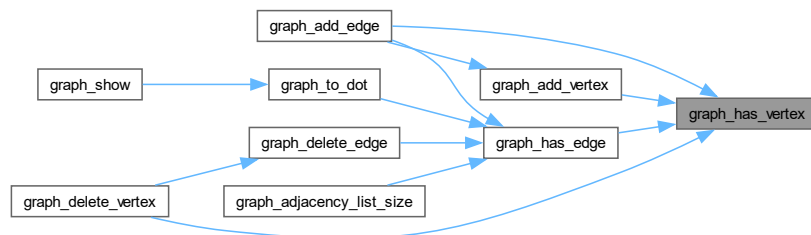
Note

- If incorrect arguments are passed, the function returns 0 (False)

Here is the call graph for this function:



Here is the caller graph for this function:



4.3.1.11 graph_initialize()

```
void graph_initialize (
    struct graph * graph )
```

Initialization of graph by zero.

Parameters

in	<i>graph</i>	Graph descriptor
----	--------------	------------------

Note

- If the graph descriptor is NULL, the function will not cause a segmentation error

4.3.1.12 graph_is_empty()

```
int graph_is_empty (
    const struct graph * graph )
```

Checking for graph emptiness.

Parameters

in	<i>graph</i>	Graph descriptor
----	--------------	------------------

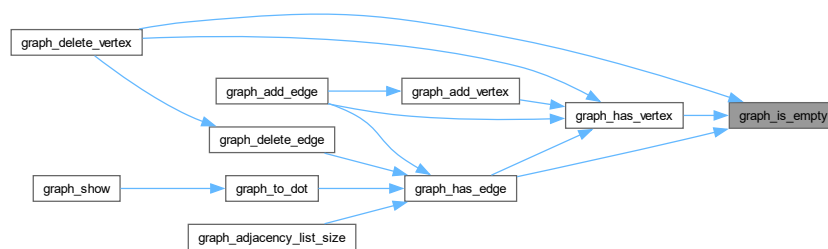
Returns

1 - True / 0 - False

Note

- If incorrect arguments are passed, the function returns 1 (True)

Here is the caller graph for this function:



4.3.1.13 graph_show()

```
graph_error_t graph_show (
    const struct graph * graph )
```

Draw graph using Graphviz and show it.

Parameters

in	<i>graph</i>	Graph descriptor
----	--------------	------------------

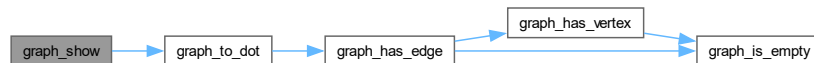
Returns

`__GRAPH_OK__`, `__GRAPH_MEM__`, `__GRAPH_INCORRECT_ARGS__`, `__GRAPH_OS_ERROR__`

Note

- Linux: the graph is demonstrated using `eog`
- Windows: the graph is demonstrated using `mspaint`
- The function creates a separate folder for temporary files and deletes it at the end of the work

Here is the call graph for this function:

4.3.1.14 `graph_to_dot()`

```

graph_error_t graph_to_dot (
    const struct graph * graph,
    const char * folder,
    const char * filename )
  
```

Creating a dot file by graph.

Parameters

in	<i>graph</i>	Graph descriptor
in	<i>folder</i>	Folder name
in	<i>filename</i>	File name

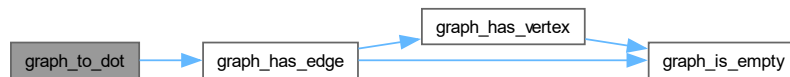
Returns

`__GRAPH_OK__`, `__GRAPH_INCORRECT_ARG__`, `__GRAPH_MEM__`, `__GRAPH_OS_ERROR__`

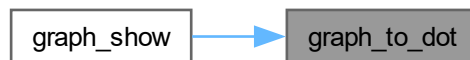
Note

- The pointer to the `folder` string can take the `NULL` value. In this case, the folder will not be created

Here is the call graph for this function:



Here is the caller graph for this function:



Index

- `_GRAPH_EMPTY__`
 - `graph.h`, 11
 - `_GRAPH_EXIST__`
 - `graph.h`, 11
 - `_GRAPH_FORBIDDEN_SEPARATORS__`
 - `graph.h`, 11
 - `_GRAPH_INCORRECT_ARG__`
 - `graph.h`, 12
 - `_GRAPH_MEM__`
 - `graph.h`, 12
 - `_GRAPH_NOT_FOUND__`
 - `graph.h`, 12
 - `_GRAPH_OK__`
 - `graph.h`, 12
 - `_GRAPH_OS_ERROR__`
 - `graph.h`, 12
 - `_STRING__`
 - `graph.h`, 12
- D:/files from internet/important/learning/github/c-modules/Graph/code/inc/graph.h, 9, 22
- D:/files from internet/important/learning/github/c-modules/Graph/code/src/graph.c, 23
- edge, 5
 - `end_vertex`, 5
 - length, 5
 - `start_vertex`, 5
- edges
 - `graph`, 7
- `edges_amount`
 - `graph`, 7
- `end_vertex`
 - edge, 5
- `graph`, 6
 - edges, 7
 - `edges_amount`, 7
 - vertices, 7
 - `vertices_amount`, 7
- `graph.c`
 - `graph_add_edge`, 24
 - `graph_add_vertex`, 25
 - `graph_adjacency_list_fill`, 26
 - `graph_adjacency_list_size`, 26
 - `graph_delete_edge`, 27
 - `graph_delete_vertex`, 28
 - `graph_dfs`, 28
 - `graph_free`, 28
 - `graph_has_edge`, 29
- `graph_has_vertex`, 30
- `graph_initialize`, 30
- `graph_is_empty`, 31
- `graph_show`, 31
- `graph_to_dot`, 32
- `graph.h`
 - `_GRAPH_EMPTY__`, 11
 - `_GRAPH_EXIST__`, 11
 - `_GRAPH_FORBIDDEN_SEPARATORS__`, 11
 - `_GRAPH_INCORRECT_ARG__`, 12
 - `_GRAPH_MEM__`, 12
 - `_GRAPH_NOT_FOUND__`, 12
 - `_GRAPH_OK__`, 12
 - `_GRAPH_OS_ERROR__`, 12
 - `_STRING__`, 12
 - `graph_add_edge`, 13
 - `graph_add_vertex`, 13
 - `graph_adjacency_list_fill`, 14
 - `graph_adjacency_list_size`, 15
 - `graph_delete_edge`, 15
 - `graph_delete_vertex`, 16
 - `graph_dfs`, 17
 - `graph_error_t`, 13
 - `graph_free`, 17
 - `graph_has_edge`, 17
 - `graph_has_vertex`, 18
 - `graph_initialize`, 19
 - `graph_is_empty`, 20
 - `graph_show`, 20
 - `graph_to_dot`, 21
- `graph_add_edge`
 - `graph.c`, 24
 - `graph.h`, 13
- `graph_add_vertex`
 - `graph.c`, 25
 - `graph.h`, 13
- `graph_adjacency_list_fill`
 - `graph.c`, 26
 - `graph.h`, 14
- `graph_adjacency_list_size`
 - `graph.c`, 26
 - `graph.h`, 15
- `graph_delete_edge`
 - `graph.c`, 27
 - `graph.h`, 15
- `graph_delete_vertex`
 - `graph.c`, 28
 - `graph.h`, 16
- `graph_dfs`

- graph.c, 28
- graph.h, 17
- graph_error_t
 - graph.h, 13
- graph_free
 - graph.c, 28
 - graph.h, 17
- graph_has_edge
 - graph.c, 29
 - graph.h, 17
- graph_has_vertex
 - graph.c, 30
 - graph.h, 18
- graph_initialize
 - graph.c, 30
 - graph.h, 19
- graph_is_empty
 - graph.c, 31
 - graph.h, 20
- graph_show
 - graph.c, 31
 - graph.h, 20
- graph_to_dot
 - graph.c, 32
 - graph.h, 21
- length
 - edge, 5
- start_vertex
 - edge, 5
- vertices
 - graph, 7
- vertices_amount
 - graph, 7