

gds

Generated by Doxygen 1.8.1.2

Sat Nov 8 2014 19:25:01

Contents

1	Module Index	1
1.1	Modules	1
2	Data Structure Index	3
2.1	Data Structures	3
3	File Index	5
3.1	File List	5
4	Module Documentation	7
4.1	Private functionality for manipulating generic datatypes	7
4.1.1	Detailed Description	8
4.1.2	Enumeration Type Documentation	8
4.1.2.1	gds_datatype	8
4.1.3	Function Documentation	8
4.1.3.1	gdt_compare	8
4.1.3.2	gdt_compare_void	8
4.1.3.3	gdt_free	9
4.1.3.4	gdt_get_value	9
4.1.3.5	gdt_reverse_compare_void	9
4.1.3.6	gdt_set_value	9
4.2	Public general generic data structures functionality	11
4.2.1	Detailed Description	11
4.2.2	Enumeration Type Documentation	11
4.2.2.1	gds_option	11
4.2.3	Function Documentation	11
4.2.3.1	gds_assert_quit	11
4.2.3.2	gds_error_quit	11
4.2.3.3	gds_strerror_quit	12
4.3	Public interface to generic list data structure	13
4.3.1	Detailed Description	13
4.3.2	Function Documentation	13

4.3.2.1	list_append	13
4.3.2.2	list_create	14
4.3.2.3	list_delete_back	14
4.3.2.4	list_delete_front	14
4.3.2.5	list_delete_index	14
4.3.2.6	list_destroy	15
4.3.2.7	list_element_at_index	15
4.3.2.8	list_find	15
4.3.2.9	list_insert	16
4.3.2.10	list_is_empty	16
4.3.2.11	list_length	16
4.3.2.12	list_prepend	16
4.3.2.13	list_set_element_at_index	17
4.4	Public interface to generic queue data structure	18
4.4.1	Detailed Description	18
4.4.2	Function Documentation	18
4.4.2.1	queue_capacity	18
4.4.2.2	queue_create	19
4.4.2.3	queue_destroy	19
4.4.2.4	queue_free_space	19
4.4.2.5	queue_is_empty	19
4.4.2.6	queue_is_full	20
4.4.2.7	queue_peek	20
4.4.2.8	queue_pop	20
4.4.2.9	queue_push	20
4.4.2.10	queue_size	21
4.5	Public interface to generic stack data structure	22
4.5.1	Detailed Description	22
4.5.2	Function Documentation	22
4.5.2.1	stack_capacity	22
4.5.2.2	stack_create	23
4.5.2.3	stack_destroy	23
4.5.2.4	stack_free_space	23
4.5.2.5	stack_is_empty	23
4.5.2.6	stack_is_full	24
4.5.2.7	stack_peek	24
4.5.2.8	stack_pop	24
4.5.2.9	stack_push	24
4.5.2.10	stack_size	25
4.6	Public interface to generic vector data structure.	26

5	Data Structure Documentation	27
5.1	gdt_generic_datatype Struct Reference	27
5.1.1	Detailed Description	27
5.1.2	Field Documentation	27
5.1.2.1	c	27
5.1.2.2	compfunc	28
5.1.2.3	d	28
5.1.2.4	data	28
5.1.2.5	i	28
5.1.2.6	l	28
5.1.2.7	ll	28
5.1.2.8	p	28
5.1.2.9	pc	28
5.1.2.10	sc	28
5.1.2.11	st	28
5.1.2.12	type	28
5.1.2.13	uc	28
5.1.2.14	ui	29
5.1.2.15	ul	29
5.1.2.16	ull	29
5.2	list Struct Reference	29
5.3	list_node Struct Reference	30
5.4	queue Struct Reference	30
5.5	stack Struct Reference	31
5.6	vector Struct Reference	32
6	File Documentation	33
6.1	include/private/gds_common.h File Reference	33
6.1.1	Detailed Description	33
6.2	include/private/gdt.h File Reference	34
6.2.1	Detailed Description	35
6.3	include/public/gds_public_types.h File Reference	35
6.3.1	Detailed Description	36
6.4	include/public/gds_util.h File Reference	37
6.4.1	Detailed Description	37
6.5	include/public/list.h File Reference	38
6.5.1	Detailed Description	39
6.6	include/public/queue.h File Reference	39
6.6.1	Detailed Description	40
6.7	include/public/stack.h File Reference	40

6.7.1 Detailed Description	42
6.8 include/public/vector.h File Reference	42
6.8.1 Detailed Description	43

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Private functionality for manipulating generic datatypes	7
Public general generic data structures functionality	11
Public interface to generic list data structure	13
Public interface to generic queue data structure	18
Public interface to generic stack data structure	22
Public interface to generic vector data structure.	26

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

gdt_generic_datatype	
Generic datatype structure	27
list	29
list_node	30
queue	30
stack	31
vector	32

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

include/private/ gds_common.h	
Common internal headers for data structures	33
include/private/ gdt.h	
Interface to generic data element functionality	34
include/public/ gds_public_types.h	
Common public types for generic data structures library	35
include/public/ gds_util.h	
Interface to general utility functions	37
include/public/ list.h	
Interface to generic list data structure	38
include/public/ queue.h	
Interface to generic queue data structure	39
include/public/ stack.h	
Interface to generic stack data structure	40
include/public/ vector.h	
Interface to generic vector data structure	42

Chapter 4

Module Documentation

4.1 Private functionality for manipulating generic datatypes

Data Structures

- struct [gdt_generic_datatype](#)
Generic datatype structure.

Typedefs

- typedef int(* [gds_cfunc](#))(const void *, const void *)
Type definition for comparison function pointer.

Enumerations

- enum [gds_datatype](#) {
 DATATYPE_CHAR, DATATYPE_UNSIGNED_CHAR, DATATYPE_SIGNED_CHAR, DATATYPE_INT,
 DATATYPE_UNSIGNED_INT, DATATYPE_LONG, DATATYPE_UNSIGNED_LONG, DATATYPE_LONG_
 LONG,
 DATATYPE_UNSIGNED_LONG_LONG, DATATYPE_SIZE_T, DATATYPE_DOUBLE, DATATYPE_STRIN-
 G,
 DATATYPE_POINTER }
Enumeration type for data element type.

Functions

- void [gdt_set_value](#) (struct [gdt_generic_datatype](#) *data, const enum [gds_datatype](#) type, [gds_cfunc](#) cfunc, va_list ap)
Sets the value of a generic datatype.
- void [gdt_get_value](#) (const struct [gdt_generic_datatype](#) *data, void *p)
Gets the value of a generic datatype.
- void [gdt_free](#) (struct [gdt_generic_datatype](#) *data)
Frees memory pointed to by a generic datatype.
- int [gdt_compare](#) (const struct [gdt_generic_datatype](#) *d1, const struct [gdt_generic_datatype](#) *d2)
Compares two generic datatypes.
- int [gdt_compare_void](#) (const void *p1, const void *p2)
Compares two generic datatypes via void pointers.
- int [gdt_reverse_compare_void](#) (const void *p1, const void *p2)
Reverse compares two generic datatypes via void pointers.

4.1.1 Detailed Description

4.1.2 Enumeration Type Documentation

4.1.2.1 enum gds_datatype

Enumeration type for data element type.

Enumerator:

DATATYPE_CHAR char
DATATYPE_UNSIGNED_CHAR unsigned char
DATATYPE_SIGNED_CHAR signed char
DATATYPE_INT int
DATATYPE_UNSIGNED_INT unsigned int
DATATYPE_LONG long
DATATYPE_UNSIGNED_LONG unsigned long
DATATYPE_LONG_LONG long long
DATATYPE_UNSIGNED_LONG_LONG unsigned long long
DATATYPE_SIZE_T size_t
DATATYPE_DOUBLE double
DATATYPE_STRING char *, string
DATATYPE_POINTER void *

4.1.3 Function Documentation

4.1.3.1 int gdt_compare (const struct gdt_generic_datatype * *d1*, const struct gdt_generic_datatype * *d2*)

Compares two generic datatypes.

Parameters

<i>d1</i>	A pointer to the first generic datatype.
<i>d2</i>	A pointer to the second generic datatype.

Return values

0	The two datatypes are equal.
-1	The first datatype is less than the second datatype.
1	The first datatype is greater than the second datatype.

4.1.3.2 int gdt_compare_void (const void * *p1*, const void * *p2*)

Compares two generic datatypes via void pointers.

This function is suitable for passing to `qsort()`.

Parameters

<i>d1</i>	A pointer to the first generic datatype.
<i>d2</i>	A pointer to the second generic datatype.

Return values

0	The two datatypes are equal.
-1	The first datatype is less than the second datatype.
1	The first datatype is greater than the second datatype.

4.1.3.3 void gdt_free (struct gdt_generic_datatype * data)

Frees memory pointed to by a generic datatype.

This function does nothing if the type of the generic datatype set by the last call to `gdt_set_value()` is neither `DATATYPE_STRING` nor `DATATYPE_POINTER`. If the type of the generic datatype is one of these values, the caller is responsible for ensuring that the last value set contains an address on which it is appropriate to call `free()`.

Parameters

<i>data</i>	A pointer to the generic datatype.
-------------	------------------------------------

4.1.3.4 void gdt_get_value (const struct gdt_generic_datatype * data, void * p)

Gets the value of a generic datatype.

Parameters

<i>data</i>	A pointer to the generic datatype.
<i>p</i>	A pointer containing the address of an object of type appropriate to the type of the generic datatype set by the last call to <code>gdt_set_value()</code> . This object will be modified to contain the value of the generic datatype.

4.1.3.5 int gdt_reverse_compare_void (const void * p1, const void * p2)

Reverse compares two generic datatypes via `void` pointers.

This function is suitable for passing to `qsort()` when the desired behavior is to sort in reverse order.

Parameters

<i>d1</i>	A pointer to the first generic datatype.
<i>d2</i>	A pointer to the second generic datatype.

Return values

0	The two datatypes are equal.
-1	The first datatype is greater than the second datatype.
1	The first datatype is less than the second datatype.

4.1.3.6 void gdt_set_value (struct gdt_generic_datatype * data, const enum gds_datatype type, gds_cfunc cfunc, va_list ap)

Sets the value of a generic datatype.

Parameters

<i>data</i>	A pointer to the generic datatype.
<i>type</i>	The type of data for the datatype to contain.
<i>cfunc</i>	A pointer to a comparison function. This is ignored for all types other than <code>DATATYPE_POINTER</code> . For <code>DATATYPE_POINTER</code> , this should contain the address of a function of type <code>int (*)(const void *, const void *)</code> if the datatype will ever need to be compared with another datatype of the same type (e.g. for finding or sorting elements within a data structure). If this functionality is not required, <code>NULL</code> can be provided.
<i>ap</i>	A <code>va_list</code> containing a single argument of the type appropriate to <code>type</code> , containing the value to which to set the generic datatype.

4.2 Public general generic data structures functionality

Enumerations

- enum `gds_option` { `GDS_RESIZABLE` = 1, `GDS_FREE_ON_DESTROY` = 2, `GDS_EXIT_ON_ERROR` = 4 }

Enumeration type for data structure options.

Functions

- void `gds_strerror_quit` (const char *msg,...)
Prints an error message with error number and exits.
- void `gds_error_quit` (const char *msg,...)
Prints an error message exits.
- void `gds_assert_quit` (const char *msg,...)
Prints an error message exits via assert().

4.2.1 Detailed Description

4.2.2 Enumeration Type Documentation

4.2.2.1 enum gds_option

Enumeration type for data structure options.

Enumerator:

`GDS_RESIZABLE` Dynamically resizes on demand

`GDS_FREE_ON_DESTROY` Automatically frees pointer members

`GDS_EXIT_ON_ERROR` Exits on error

4.2.3 Function Documentation

4.2.3.1 void gds_assert_quit (const char * msg, ...)

Prints an error message exits via assert().

This function will do nothing if `NDEBUG` is defined. Otherwise, it behaves in a manner identical to `gds_error_quit()` except it terminates via `assert()`, rather than `exit()`.

Parameters

<code>msg</code>	The format string for the message to print. Format specifiers are the same as the <code>printf()</code> family of functions.
<code>...</code>	Any arguments to the format string.

4.2.3.2 void gds_error_quit (const char * msg, ...)

Prints an error message exits.

Parameters

<i>msg</i>	The format string for the message to print. Format specifiers are the same as the <code>printf()</code> family of functions.
...	Any arguments to the format string.

4.2.3.3 `void gds_strerror_quit (const char * msg, ...)`

Prints an error message with error number and exits.

This function can be called to print an error message and quit following a function which has indicated failure and has set `errno`. A message containing the error number and a text representation of that error will be printed, following by the message supplied to the function.

Parameters

<i>msg</i>	The format string for the message to print. Format specifiers are the same as the <code>printf()</code> family of functions.
...	Any arguments to the format string.

4.3 Public interface to generic list data structure

Typedefs

- typedef struct [list](#) * [List](#)
Opaque list type definition.

Functions

- [List](#) [list_create](#) (const enum [gds_datatype](#) type, const int opts,...)
Creates a new list.
- void [list_destroy](#) ([List](#) list)
Destroys a list.
- bool [list_append](#) ([List](#) list,...)
Appends a value to the back of a list.
- bool [list_prepend](#) ([List](#) list,...)
Prepends a value to the front of a list.
- bool [list_insert](#) ([List](#) list, const size_t index,...)
Inserts a value into a list.
- bool [list_delete_front](#) ([List](#) list)
Deletes the value at the front of the list.
- bool [list_delete_back](#) ([List](#) list)
Deletes the value at the back of the list.
- bool [list_delete_index](#) ([List](#) list, const size_t index)
Deletes the value at the specified index of the list.
- bool [list_element_at_index](#) ([List](#) list, const size_t index, void *p)
Gets the value at the specified index of the list.
- bool [list_set_element_at_index](#) ([List](#) list, const size_t index,...)
Sets the value at the specified index of the list.
- bool [list_find](#) ([List](#) list, size_t *index,...)
Tests if a value is contained in a list.
- bool [list_is_empty](#) ([List](#) list)
Tests if a list is empty.
- size_t [list_length](#) ([List](#) list)
Returns the length of a list.

4.3.1 Detailed Description

4.3.2 Function Documentation

4.3.2.1 bool list_append (List list, ...)

Appends a value to the back of a list.

Parameters

<i>list</i>	A pointer to the list.
...	The value to append to the end of the list. This should be of a type appropriate to the type set when creating the list.

Return values

<i>true</i>	Success
<i>false</i>	Failure, dynamic memory allocation failed.

4.3.2.2 List `list.create (const enum gds_datatype type, const int opts, ...)` [read]

Creates a new list.

Parameters

<i>type</i>	The datatype for the list.
<i>opts</i>	The following options can be OR'd together: <code>GDS_FREE_ON_DESTROY</code> to automatically <code>free()</code> pointer members when they are deleted or when the list is destroyed; <code>GDS_EXIT_ON_ERROR</code> to print a message to the standard error stream and <code>exit()</code> , rather than returning a failure status.
<i>...</i>	If <i>type</i> is <code>DATATYPE_POINTER</code> , this argument should be a pointer to a comparison function. In all other cases, this argument is not required, and will be ignored if it is provided.

Return values

<i>NULL</i>	List creation failed.
<i>non-NULL</i>	A pointer to the new list.

4.3.2.3 `bool list.delete.back (List list)`

Deletes the value at the back of the list.

Parameters

<i>list</i>	A pointer to the list.
-------------	------------------------

Return values

<i>true</i>	Success
<i>false</i>	Failure, dynamic memory allocation failed.

4.3.2.4 `bool list.delete.front (List list)`

Deletes the value at the front of the list.

Parameters

<i>list</i>	A pointer to the list.
-------------	------------------------

Return values

<i>true</i>	Success
<i>false</i>	Failure, dynamic memory allocation failed.

4.3.2.5 `bool list.delete.index (List list, const size_t index)`

Deletes the value at the specified index of the list.

Parameters

<i>list</i>	A pointer to the list.
<i>index</i>	The index of the value to delete.

Return values

<i>true</i>	Success
<i>false</i>	Failure, dynamic memory allocation failed or index was out of range.

4.3.2.6 void list_destroy (List list)

Destroys a list.

If the `GDS_FREE_ON_DESTROY` option was specified when creating the list, any pointer values still in the list will be `free()`d prior to destruction.

Parameters

<i>list</i>	A pointer to the list.
-------------	------------------------

4.3.2.7 bool list_element_at_index (List list, const size_t index, void * p)

Gets the value at the specified index of the list.

Parameters

<i>list</i>	A pointer to the list.
<i>index</i>	The index of the value to get.
<i>p</i>	A pointer to an object of a type appropriate to the type set when creating the list. The object at this address will be modified to contain the value at the specified index.

Return values

<i>true</i>	Success
<i>false</i>	Failure, index was out of range.

4.3.2.8 bool list_find (List list, size_t * index, ...)

Tests if a value is contained in a list.

Parameters

<i>list</i>	A pointer to the list.
<i>index</i>	A pointer to a <code>size_t</code> object which, if the value is contained within the list, will be modified to contain the index of the first occurrence of that value in the list.
<i>...</i>	The value for which to search. This should be of a type appropriate to the type set when creating the list.

Return values

<i>true</i>	The value was found in the list
<i>false</i>	The value was not found in the list

4.3.2.9 `bool list_insert (List list, const size_t index, ...)`

Inserts a value into a list.

Parameters

<i>list</i>	A pointer to the list.
<i>index</i>	The index at which to insert the value.
<i>...</i>	The value to insert into the list. This should be of a type appropriate to the type set when creating the list.

Return values

<i>true</i>	Success
<i>false</i>	Failure, dynamic memory allocation failed or index was out of range.

4.3.2.10 `bool list_is_empty (List list)`

Tests if a list is empty.

Parameters

<i>list</i>	A pointer to the list.
-------------	------------------------

Return values

<i>true</i>	The list is empty
<i>false</i>	The list is not empty

4.3.2.11 `size_t list_length (List list)`

Returns the length of a list.

The length of the list is equivalent to the number of values it contains.

Parameters

<i>list</i>	A pointer to the list.
-------------	------------------------

Returns

The length of the list.

4.3.2.12 `bool list_prepend (List list, ...)`

Prepends a value to the front of a list.

Parameters

<i>list</i>	A pointer to the list.
<i>...</i>	The value to prepend to the start of the list. This should be of a type appropriate to the type set when creating the list.

Return values

<i>true</i>	Success
<i>false</i>	Failure, dynamic memory allocation failed.

4.3.2.13 `bool list_set_element_at_index (List list, const size_t index, ...)`

Sets the value at the specified index of the list.

Parameters

<i>list</i>	A pointer to the list.
<i>index</i>	The index of the value to set.
...	The value to which to set the specified index of the list. This should be of a type appropriate to the type set when creating the list.

Return values

<i>true</i>	Success
<i>false</i>	Failure, index was out of range.

4.4 Public interface to generic queue data structure

Typedefs

- typedef struct `queue` * `Queue`
Opaque queue type definition.

Functions

- `Queue queue_create` (const size_t capacity, const enum `gds_datatype` type, const int opts)
Creates a new queue.
- void `queue_destroy` (`Queue queue`)
Destroys a queue.
- bool `queue_push` (`Queue queue`,...)
Pushes a value onto the queue.
- bool `queue_pop` (`Queue queue`, void *p)
Pops a value from the queue.
- bool `queue_peek` (`Queue queue`, void *p)
Peeks at the top value of the queue.
- bool `queue_is_full` (`Queue queue`)
Checks whether a queue is full.
- bool `queue_is_empty` (`Queue queue`)
Checks whether a queue is empty.
- size_t `queue_capacity` (`Queue queue`)
Retrieves the current capacity of a queue.
- size_t `queue_size` (`Queue queue`)
Retrieves the current size of a queue.
- size_t `queue_free_space` (`Queue queue`)
Retrieves the free space on a queue.

4.4.1 Detailed Description

A queue is a first-in-first-out (FIFO) data structure. Two fundamental operations are possible. A value can be *pushed* onto the queue, and a value can be *popped* from the queue. By virtue of being a FIFO data structure, pushing and popping happen at opposite ends of the queue. In other words, the value popped will be the first item pushed onto the queue that has not already been popped from it.

4.4.2 Function Documentation

4.4.2.1 size_t queue_capacity (`Queue queue`)

Retrieves the current capacity of a queue.

This value can change dynamically if the `GDS_RESIZABLE` option was specified when creating the queue.

Parameters

<code>queue</code>	A pointer to the queue.
--------------------	-------------------------

Returns

The capacity of the queue.

4.4.2.2 Queue queue_create (const size_t capacity, const enum gds_datatype type, const int opts) [read]

Creates a new queue.

Parameters

<i>capacity</i>	The initial capacity of the queue.
<i>type</i>	The datatype for the queue.
<i>opts</i>	The following options can be OR'd together: GDS_FREE_ON_DESTROY to automatically free() pointer members when they are deleted or when the queue is destroyed; GDS_EXIT_ON_ERROR to print a message to the standard error stream and exit(), rather than returning a failure status.

Return values

<i>NULL</i>	Queue creation failed.
<i>non-NULL</i>	A pointer to the new queue.

4.4.2.3 void queue_destroy (Queue queue)

Destroys a queue.

If the GDS_FREE_ON_DESTROY option was specified when creating the queue, any pointer values still in the queue will be free()d prior to destruction.

Parameters

<i>stack</i>	A pointer to the queue.
--------------	-------------------------

4.4.2.4 size_t queue_free_space (Queue queue)

Retrieves the free space on a queue.

The free space on a queue is equivalent to the capacity of the queue less the size of the queue.

Parameters

<i>queue</i>	A pointer to the queue.
--------------	-------------------------

Returns

The free space on the queue.

4.4.2.5 bool queue_is_empty (Queue queue)

Checks whether a queue is empty.

Parameters

<i>queue</i>	A pointer to the queue.
--------------	-------------------------

Return values

<i>true</i>	Queue is empty
<i>false</i>	Queue is not empty

4.4.2.6 `bool queue_is_full (Queue queue)`

Checks whether a queue is full.

Parameters

<i>queue</i>	A pointer to the queue.
--------------	-------------------------

Return values

<i>true</i>	Queue is full
<i>false</i>	Queue is not full

4.4.2.7 `bool queue_peek (Queue queue, void * p)`

Peeks at the top value of the queue.

This function retrieves the value which would be popped from the queue, without actually popping it.

Parameters

<i>list</i>	A pointer to the queue.
<i>p</i>	A pointer to an object of a type appropriate to the type set when creating the queue. The object at this address will be modified to contain the value at the top of the queue.

Return values

<i>true</i>	Success
<i>false</i>	Failure, queue is empty.

4.4.2.8 `bool queue_pop (Queue queue, void * p)`

Pops a value from the queue.

Parameters

<i>list</i>	A pointer to the queue.
<i>p</i>	A pointer to an object of a type appropriate to the type set when creating the queue. The object at this address will be modified to contain the value popped from the queue.

Return values

<i>true</i>	Success
<i>false</i>	Failure, queue is empty.

4.4.2.9 `bool queue_push (Queue queue, ...)`

Pushes a value onto the queue.

Parameters

<i>list</i>	A pointer to the queue.
<i>...</i>	The value to push onto the queue. This should be of a type appropriate to the type set when creating the queue.

Return values

<i>true</i>	Success
<i>false</i>	Failure, either because the queue is full or, if the <code>GDS_RESIZABLE</code> option was specified when creating the queue, because dynamic memory reallocation failed.

4.4.2.10 `size_t queue_size (Queue queue)`

Retrieves the current size of a queue.

The size of the queue is equivalent to the number of values currently in it.

Parameters

<i>queue</i>	A pointer to the queue.
--------------	-------------------------

Returns

The size of the queue.

4.5 Public interface to generic stack data structure

Typedefs

- typedef struct `stack` * `Stack`
Opaque stack type definition.

Functions

- `Stack stack_create` (const `size_t` capacity, const enum `gds_datatype` type, const int opts)
Creates a new stack.
- void `stack_destroy` (`Stack stack`)
Destroys a stack.
- bool `stack_push` (`Stack stack`,...)
Pushes a value onto the stack.
- bool `stack_pop` (`Stack stack`, void *p)
Pops a value from the stack.
- bool `stack_peek` (`Stack stack`, void *p)
Peeks at the top value of the stack.
- bool `stack_is_full` (`Stack stack`)
Checks whether a stack is full.
- bool `stack_is_empty` (`Stack stack`)
Checks whether a stack is empty.
- `size_t stack_capacity` (`Stack stack`)
Retrieves the current capacity of a stack.
- `size_t stack_size` (`Stack stack`)
Retrieves the current size of a stack.
- `size_t stack_free_space` (`Stack stack`)
Retrieves the free space on a stack.

4.5.1 Detailed Description

A stack is a last-in-first-out (LIFO) data structure. Two fundamental operations are possible. A value can be *pushed* onto the stack, and a value can be *popped* from the stack. By virtue of being a LIFO data structure, pushing and popping happen at the same end of the stack. In other words, the value popped will be the last item pushed onto the stack that has not already been popped from it.

4.5.2 Function Documentation

4.5.2.1 `size_t stack_capacity (Stack stack)`

Retrieves the current capacity of a stack.

This value can change dynamically if the `GDS_RESIZABLE` option was specified when creating the stack.

Parameters

<code>stack</code>	A pointer to the stack.
--------------------	-------------------------

Returns

The capacity of the stack.

4.5.2.2 Stack `stack_create (const size_t capacity, const enum gds_datatype type, const int opts)` [read]

Creates a new stack.

Parameters

<i>capacity</i>	The initial capacity of the stack.
<i>type</i>	The datatype for the stack.
<i>opts</i>	The following options can be OR'd together: <code>GDS_FREE_ON_DESTROY</code> to automatically <code>free()</code> pointer members when they are deleted or when the stack is destroyed; <code>GDS_EXIT_ON_ERROR</code> to print a message to the standard error stream and <code>exit()</code> , rather than returning a failure status.

Return values

<i>NULL</i>	Stack creation failed.
<i>non-NULL</i>	A pointer to the new stack.

4.5.2.3 void `stack_destroy (Stack stack)`

Destroys a stack.

If the `GDS_FREE_ON_DESTROY` option was specified when creating the stack, any pointer values still in the stack will be `free()`d prior to destruction.

Parameters

<i>stack</i>	A pointer to the stack.
--------------	-------------------------

4.5.2.4 size_t `stack_free_space (Stack stack)`

Retrieves the free space on a stack.

The free space on a stack is equivalent to the capacity of the stack less the size of the stack.

Parameters

<i>stack</i>	A pointer to the stack.
--------------	-------------------------

Returns

The free space on the stack.

4.5.2.5 bool `stack_is_empty (Stack stack)`

Checks whether a stack is empty.

Parameters

<i>stack</i>	A pointer to the stack.
--------------	-------------------------

Return values

<i>true</i>	Stack is empty
<i>false</i>	Stack is not empty

4.5.2.6 `bool stack_is_full (Stack stack)`

Checks whether a stack is full.

Parameters

<i>stack</i>	A pointer to the stack.
--------------	-------------------------

Return values

<i>true</i>	Stack is full
<i>false</i>	Stack is not full

4.5.2.7 `bool stack_peek (Stack stack, void * p)`

Peeks at the top value of the stack.

This function retrieves the value which would be popped from the stack, without actually popping it.

Parameters

<i>list</i>	A pointer to the stack.
<i>p</i>	A pointer to an object of a type appropriate to the type set when creating the stack. The object at this address will be modified to contain the value at the top of the stack.

Return values

<i>true</i>	Success
<i>false</i>	Failure, stack is empty.

4.5.2.8 `bool stack_pop (Stack stack, void * p)`

Pops a value from the stack.

Parameters

<i>list</i>	A pointer to the stack.
<i>p</i>	A pointer to an object of a type appropriate to the type set when creating the stack. The object at this address will be modified to contain the value popped from the stack.

Return values

<i>true</i>	Success
<i>false</i>	Failure, stack is empty.

4.5.2.9 `bool stack_push (Stack stack, ...)`

Pushes a value onto the stack.

Parameters

<i>list</i>	A pointer to the stack.
<i>...</i>	The value to push onto the stack. This should be of a type appropriate to the type set when creating the stack.

Return values

<i>true</i>	Success
<i>false</i>	Failure, either because the stack is full or, if the <code>GDS_RESIZABLE</code> option was specified when creating the stack, because dynamic memory reallocation failed.

4.5.2.10 `size_t stack_size (Stack stack)`

Retrieves the current size of a stack.

The size of the stack is equivalent to the number of values currently in it.

Parameters

<i>stack</i>	A pointer to the stack.
--------------	-------------------------

Returns

The size of the stack.

4.6 Public interface to generic vector data structure.

Chapter 5

Data Structure Documentation

5.1 gdt_generic_datatype Struct Reference

Generic datatype structure.

```
#include <gdt.h>
```

Data Fields

- enum [gds_datatype](#) type
 - [gds_cfunc](#) compfunc
 - union {
 - char [c](#)
 - unsigned char [uc](#)
 - signed char [sc](#)
 - int [i](#)
 - unsigned int [ui](#)
 - long [l](#)
 - unsigned long [ul](#)
 - long long int [ll](#)
 - unsigned long long int [ull](#)
 - size_t [st](#)
 - double [d](#)
 - char * [pc](#)
 - void * [p](#)
- } [data](#)

5.1.1 Detailed Description

Generic datatype structure.

5.1.2 Field Documentation

5.1.2.1 char gdt_generic_datatype::c

char

5.1.2.2 gds_cfunc gdt_generic_datatype::compfunc

Comparison function pointer

5.1.2.3 double gdt_generic_datatype::d

double

5.1.2.4 union { ... } gdt_generic_datatype::data

Data union

5.1.2.5 int gdt_generic_datatype::i

int

5.1.2.6 long gdt_generic_datatype::l

long

5.1.2.7 long long int gdt_generic_datatype::ll

long long

5.1.2.8 void* gdt_generic_datatype::p

void *

5.1.2.9 char* gdt_generic_datatype::pc

char *, string

5.1.2.10 signed char gdt_generic_datatype::sc

signed char

5.1.2.11 size_t gdt_generic_datatype::st

size_t

5.1.2.12 enum gds_datatype gdt_generic_datatype::type

Data type

5.1.2.13 unsigned char gdt_generic_datatype::uc

unsigned char

5.1.2.14 unsigned int gdt_generic_datatype::ui

unsigned int

5.1.2.15 unsigned long gdt_generic_datatype::ul

unsigned long

5.1.2.16 unsigned long long int gdt_generic_datatype::ull

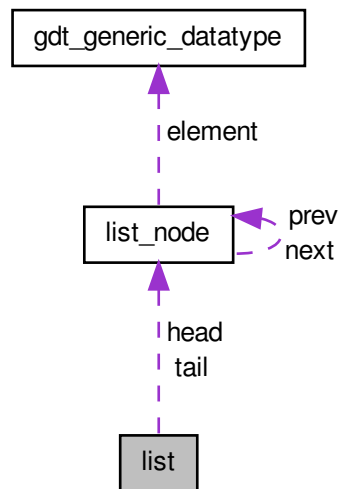
unsigned long long

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

- [include/private/gdt.h](#)

5.2 list Struct Reference

Collaboration diagram for list:



Data Fields

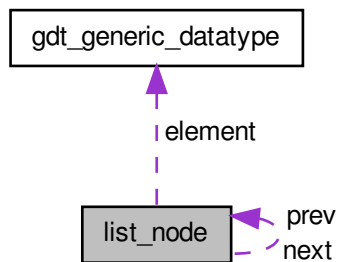
- `size_t` **length**
- enum [gds_datatype](#) **type**
- [gds_cfunc](#) **compfunc**
- struct [list_node](#) * **head**
- struct [list_node](#) * **tail**
- bool **free_on_destroy**
- bool **exit_on_error**

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

- src/list.c

5.3 list_node Struct Reference

Collaboration diagram for list_node:



Data Fields

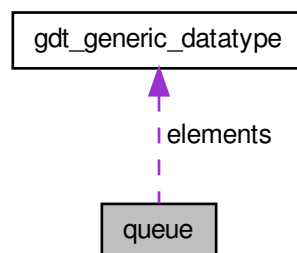
- struct `gdt_generic_datatype` `element`
- struct `list_node` * `prev`
- struct `list_node` * `next`

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

- src/list.c

5.4 queue Struct Reference

Collaboration diagram for queue:



Data Fields

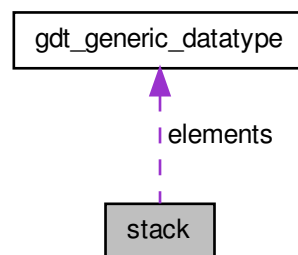
- `size_t front`
- `size_t back`
- `size_t capacity`
- `size_t size`
- enum `gds_datatype type`
- struct `gdt_generic_datatype * elements`
- `bool resizable`
- `bool free_on_destroy`
- `bool exit_on_error`

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

- `src/queue.c`

5.5 stack Struct Reference

Collaboration diagram for stack:



Data Fields

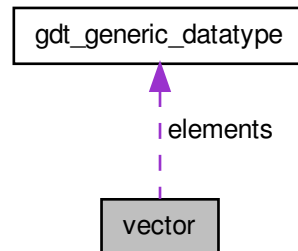
- `size_t top`
- `size_t capacity`
- enum `gds_datatype type`
- struct `gdt_generic_datatype * elements`
- `bool resizable`
- `bool free_on_destroy`
- `bool exit_on_error`

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

- `src/stack.c`

5.6 vector Struct Reference

Collaboration diagram for vector:



Data Fields

- `size_t` **length**
- `size_t` **capacity**
- `enum` `gds_datatype` **type**
- `struct` `gdt_generic_datatype` * **elements**
- `int`(* **compfunc**)(const void *, const void *)
- `bool` **free_on_destroy**
- `bool` **exit_on_error**

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

- `src/vector.c`

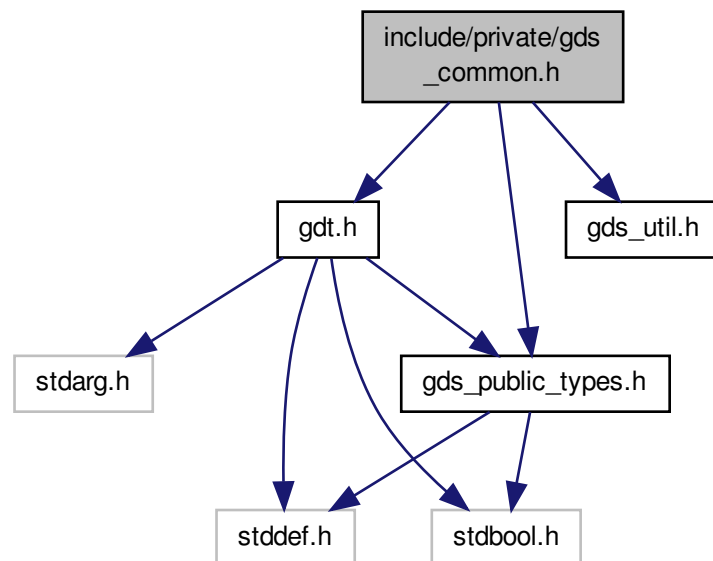
Chapter 6

File Documentation

6.1 include/private/gds_common.h File Reference

Common internal headers for data structures.

```
#include "gds_public_types.h"  
#include "gdt.h"  
#include "gds_util.h"  
Include dependency graph for gds_common.h:
```



6.1.1 Detailed Description

Common internal headers for data structures.

Author

Paul Griffiths

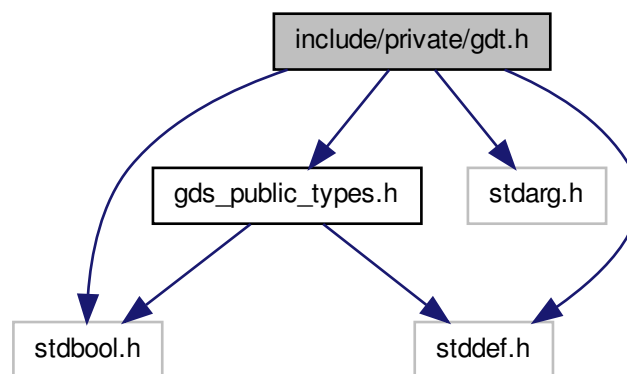
Copyright

Copyright 2014 Paul Griffiths. Distributed under the terms of the GNU General Public License. <http://www.gnu.org/licenses/>

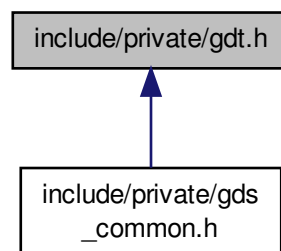
6.2 include/private/gdt.h File Reference

Interface to generic data element functionality.

```
#include <stdbool.h>
#include <stddef.h>
#include <stdarg.h>
#include "gds_public_types.h"
Include dependency graph for gdt.h:
```



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



Data Structures

- struct [gdt_generic_datatype](#)

Generic datatype structure.

Functions

- void [gdt_set_value](#) (struct [gdt_generic_datatype](#) *data, const enum [gds_datatype](#) type, [gds_cfunc](#) cfunc, va_list ap)

Sets the value of a generic datatype.

- void [gdt_get_value](#) (const struct [gdt_generic_datatype](#) *data, void *p)

Gets the value of a generic datatype.

- void [gdt_free](#) (struct [gdt_generic_datatype](#) *data)

Frees memory pointed to by a generic datatype.

- int [gdt_compare](#) (const struct [gdt_generic_datatype](#) *d1, const struct [gdt_generic_datatype](#) *d2)

Compares two generic datatypes.

- int [gdt_compare_void](#) (const void *p1, const void *p2)

Compares two generic datatypes via void pointers.

- int [gdt_reverse_compare_void](#) (const void *p1, const void *p2)

Reverse compares two generic datatypes via void pointers.

6.2.1 Detailed Description

Interface to generic data element functionality.

Author

Paul Griffiths

Copyright

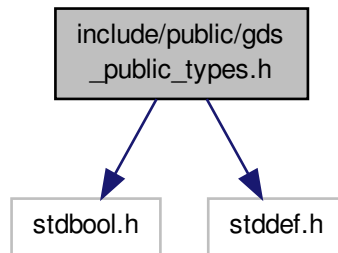
Copyright 2014 Paul Griffiths. Distributed under the terms of the GNU General Public License. <http://www.gnu.org/licenses/>

6.3 include/public/gds_public_types.h File Reference

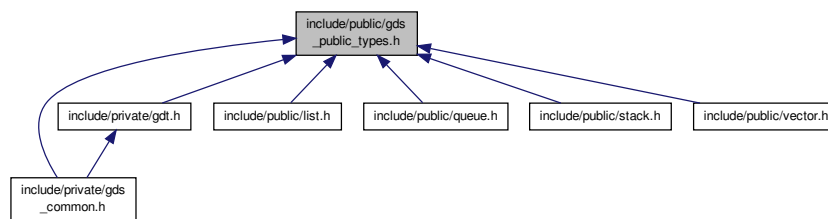
Common public types for generic data structures library.

```
#include <stdbool.h>
#include <stddef.h>
```

Include dependency graph for gds_public_types.h:



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



Typedefs

- typedef int(* [gds_cfunc](#))(const void *, const void *)

Type definition for comparison function pointer.

Enumerations

- enum [gds_option](#) { [GDS_RESIZABLE](#) = 1, [GDS_FREE_ON_DESTROY](#) = 2, [GDS_EXIT_ON_ERROR](#) = 4 }

Enumeration type for data structure options.

- enum [gds_datatype](#) { [DATATYPE_CHAR](#), [DATATYPE_UNSIGNED_CHAR](#), [DATATYPE_SIGNED_CHAR](#), [DATATYPE_INT](#), [DATATYPE_UNSIGNED_INT](#), [DATATYPE_LONG](#), [DATATYPE_UNSIGNED_LONG](#), [DATATYPE_LONG_LONG](#), [DATATYPE_UNSIGNED_LONG_LONG](#), [DATATYPE_SIZE_T](#), [DATATYPE_DOUBLE](#), [DATATYPE_STRING](#), [DATATYPE_POINTER](#) }

Enumeration type for data element type.

6.3.1 Detailed Description

Common public types for generic data structures library.

Author

Paul Griffiths

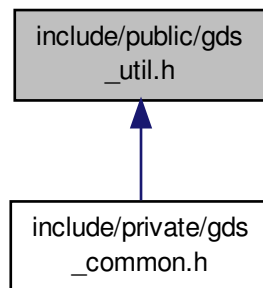
Copyright

Copyright 2014 Paul Griffiths. Distributed under the terms of the GNU General Public License. <http://www.gnu.org/licenses/>

6.4 include/public/gds_util.h File Reference

Interface to general utility functions.

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

**Functions**

- void [gds_strerror_quit](#) (const char *msg,...)
Prints an error message with error number and exits.
- void [gds_error_quit](#) (const char *msg,...)
Prints an error message exits.
- void [gds_assert_quit](#) (const char *msg,...)
Prints an error message exits via assert().

6.4.1 Detailed Description

Interface to general utility functions.

Author

Paul Griffiths

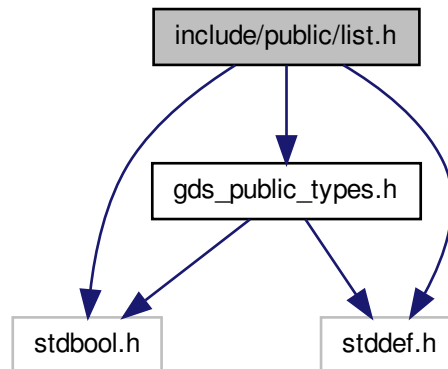
Copyright

Copyright 2014 Paul Griffiths. Distributed under the terms of the GNU General Public License. <http://www.gnu.org/licenses/>

6.5 include/public/list.h File Reference

Interface to generic list data structure.

```
#include <stdbool.h>
#include <stddef.h>
#include "gds_public_types.h"
Include dependency graph for list.h:
```



Typedefs

- typedef struct `list` * `List`
Opaque list type definition.

Functions

- `List list_create` (const enum `gds_datatype` type, const int opts,...)
Creates a new list.
- void `list_destroy` (`List` list)
Destroys a list.
- bool `list_append` (`List` list,...)
Appends a value to the back of a list.
- bool `list_prepend` (`List` list,...)
Prepends a value to the front of a list.
- bool `list_insert` (`List` list, const `size_t` index,...)
Inserts a value into a list.
- bool `list_delete_front` (`List` list)
Deletes the value at the front of the list.
- bool `list_delete_back` (`List` list)
Deletes the value at the back of the list.
- bool `list_delete_index` (`List` list, const `size_t` index)
Deletes the value at the specified index of the list.
- bool `list_element_at_index` (`List` list, const `size_t` index, void *p)
Gets the value at the specified index of the list.

- bool `list_set_element_at_index` (`List list`, `const size_t index`,...)
Sets the value at the specified index of the list.
- bool `list_find` (`List list`, `size_t *index`,...)
Tests if a value is contained in a list.
- bool `list_is_empty` (`List list`)
Tests if a list is empty.
- `size_t list_length` (`List list`)
Returns the length of a list.

6.5.1 Detailed Description

Interface to generic list data structure. The list is implemented as a double-ended, double-linked list.

Author

Paul Griffiths

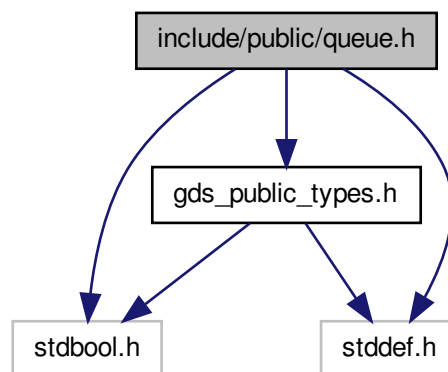
Copyright

Copyright 2014 Paul Griffiths. Distributed under the terms of the GNU General Public License. <http://www.gnu.org/licenses/>

6.6 include/public/queue.h File Reference

Interface to generic queue data structure.

```
#include <stdbool.h>
#include <stddef.h>
#include "gds_public_types.h"
Include dependency graph for queue.h:
```



Typedefs

- typedef struct `queue` * `Queue`
Opaque queue type definition.

Functions

- `Queue queue_create` (const size_t capacity, const enum `gds_datatype` type, const int opts)

Creates a new queue.

- void `queue_destroy` (Queue queue)

Destroys a queue.

- bool `queue_push` (Queue queue,...)

Pushes a value onto the queue.

- bool `queue_pop` (Queue queue, void *p)

Pops a value from the queue.

- bool `queue_peek` (Queue queue, void *p)

Peeks at the top value of the queue.

- bool `queue_is_full` (Queue queue)

Checks whether a queue is full.

- bool `queue_is_empty` (Queue queue)

Checks whether a queue is empty.

- size_t `queue_capacity` (Queue queue)

Retrieves the current capacity of a queue.

- size_t `queue_size` (Queue queue)

Retrieves the current size of a queue.

- size_t `queue_free_space` (Queue queue)

Retrieves the free space on a queue.

6.6.1 Detailed Description

Interface to generic queue data structure.

Author

Paul Griffiths

Copyright

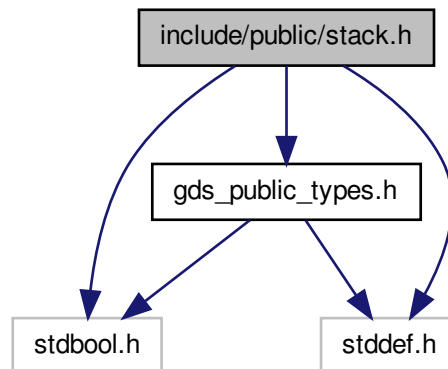
Copyright 2014 Paul Griffiths. Distributed under the terms of the GNU General Public License. <http://www.gnu.org/licenses/>

6.7 include/public/stack.h File Reference

Interface to generic stack data structure.

```
#include <stdbool.h>
#include <stddef.h>
#include "gds_public_types.h"
```

Include dependency graph for stack.h:



Typedefs

- typedef struct [stack](#) * [Stack](#)
Opaque stack type definition.

Functions

- [Stack](#) [stack_create](#) (const [size_t](#) capacity, const enum [gds_datatype](#) type, const int opts)
Creates a new stack.
- void [stack_destroy](#) ([Stack](#) stack)
Destroys a stack.
- bool [stack_push](#) ([Stack](#) stack,...)
Pushes a value onto the stack.
- bool [stack_pop](#) ([Stack](#) stack, void *p)
Pops a value from the stack.
- bool [stack_peek](#) ([Stack](#) stack, void *p)
Peeks at the top value of the stack.
- bool [stack_is_full](#) ([Stack](#) stack)
Checks whether a stack is full.
- bool [stack_is_empty](#) ([Stack](#) stack)
Checks whether a stack is empty.
- [size_t](#) [stack_capacity](#) ([Stack](#) stack)
Retrieves the current capacity of a stack.
- [size_t](#) [stack_size](#) ([Stack](#) stack)
Retrieves the current size of a stack.
- [size_t](#) [stack_free_space](#) ([Stack](#) stack)
Retrieves the free space on a stack.

6.7.1 Detailed Description

Interface to generic stack data structure.

Author

Paul Griffiths

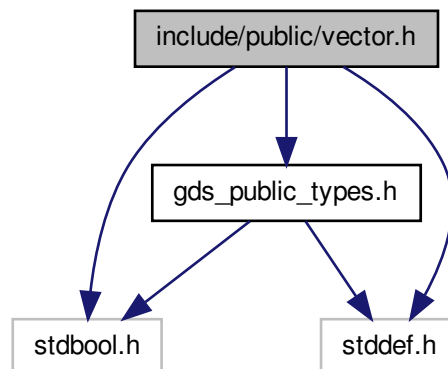
Copyright

Copyright 2014 Paul Griffiths. Distributed under the terms of the GNU General Public License. <http://www.gnu.org/licenses/>

6.8 include/public/vector.h File Reference

Interface to generic vector data structure.

```
#include <stdbool.h>
#include <stddef.h>
#include "gds_public_types.h"
Include dependency graph for vector.h:
```



Typedefs

- typedef struct `vector` * `Vector`

Functions

- `Vector` `vector_create` (const size_t capacity, const enum `gds_datatype` type, const int opts,...)
- void `vector_destroy` (`Vector` `vector`)
- bool `vector_append` (`Vector` `vector`,...)
- bool `vector_prepend` (`Vector` `vector`,...)
- bool `vector_insert` (`Vector` `vector`, const size_t index,...)
- bool `vector_delete_index` (`Vector` `vector`, const size_t index)
- bool `vector_delete_front` (`Vector` `vector`)

- bool **vector_delete_back** ([Vector vector](#))
- bool **vector_element_at_index** ([Vector vector](#), const size_t index, void *p)
- bool **vector_set_element_at_index** ([Vector vector](#), const size_t index,...)
- bool **vector_find** ([Vector vector](#), size_t *index,...)
- void **vector_sort** ([Vector vector](#))
- void **vector_reverse_sort** ([Vector vector](#))
- bool **vector_is_empty** ([Vector vector](#))
- size_t **vector_length** ([Vector vector](#))
- size_t **vector_capacity** ([Vector vector](#))
- size_t **vector_free_space** ([Vector vector](#))

6.8.1 Detailed Description

Interface to generic vector data structure.

Author

Paul Griffiths

Copyright

Copyright 2014 Paul Griffiths. Distributed under the terms of the GNU General Public License. <http://www.gnu.org/licenses/>

Index

- c
 - gdt_generic_datatype, [27](#)
- compfunc
 - gdt_generic_datatype, [27](#)
- d
 - gdt_generic_datatype, [28](#)
 - DATATYPE_CHAR
 - Private functionality for manipulating generic datatypes, [8](#)
 - DATATYPE_DOUBLE
 - Private functionality for manipulating generic datatypes, [8](#)
 - DATATYPE_INT
 - Private functionality for manipulating generic datatypes, [8](#)
 - DATATYPE_LONG
 - Private functionality for manipulating generic datatypes, [8](#)
 - DATATYPE_LONG_LONG
 - Private functionality for manipulating generic datatypes, [8](#)
 - DATATYPE_POINTER
 - Private functionality for manipulating generic datatypes, [8](#)
 - DATATYPE_SIGNED_CHAR
 - Private functionality for manipulating generic datatypes, [8](#)
 - DATATYPE_SIZE_T
 - Private functionality for manipulating generic datatypes, [8](#)
 - DATATYPE_STRING
 - Private functionality for manipulating generic datatypes, [8](#)
 - DATATYPE_UNSIGNED_CHAR
 - Private functionality for manipulating generic datatypes, [8](#)
 - DATATYPE_UNSIGNED_INT
 - Private functionality for manipulating generic datatypes, [8](#)
 - DATATYPE_UNSIGNED_LONG
 - Private functionality for manipulating generic datatypes, [8](#)
 - DATATYPE_UNSIGNED_LONG_LONG
 - Private functionality for manipulating generic datatypes, [8](#)
 - data
 - gdt_generic_datatype, [28](#)
 - GDS_EXIT_ON_ERROR
 - Public general generic data structures functionality, [11](#)
 - GDS_FREE_ON_DESTROY
 - Public general generic data structures functionality, [11](#)
 - GDS_RESIZABLE
 - Public general generic data structures functionality, [11](#)
 - gds_assert_quit
 - Public general generic data structures functionality, [11](#)
 - gds_datatype
 - Private functionality for manipulating generic datatypes, [8](#)
 - gds_error_quit
 - Public general generic data structures functionality, [11](#)
 - gds_option
 - Public general generic data structures functionality, [11](#)
 - gds_strerror_quit
 - Public general generic data structures functionality, [12](#)
 - gdt_compare
 - Private functionality for manipulating generic datatypes, [8](#)
 - gdt_compare_void
 - Private functionality for manipulating generic datatypes, [8](#)
 - gdt_free
 - Private functionality for manipulating generic datatypes, [9](#)
 - gdt_generic_datatype, [27](#)
 - c, [27](#)
 - compfunc, [27](#)
 - d, [28](#)
 - data, [28](#)
 - i, [28](#)
 - l, [28](#)
 - ll, [28](#)
 - p, [28](#)
 - pc, [28](#)
 - sc, [28](#)
 - st, [28](#)
 - type, [28](#)
 - uc, [28](#)
 - ui, [28](#)
 - ul, [29](#)
 - ull, [29](#)

- gdt_get_value
 - Private functionality for manipulating generic datatypes, [9](#)
- gdt_reverse_compare_void
 - Private functionality for manipulating generic datatypes, [9](#)
- gdt_set_value
 - Private functionality for manipulating generic datatypes, [9](#)
- i
 - gdt_generic_datatype, [28](#)
 - include/private/gds_common.h, [33](#)
 - include/private/gdt.h, [34](#)
 - include/public/gds_public_types.h, [35](#)
 - include/public/gds_util.h, [37](#)
 - include/public/list.h, [38](#)
 - include/public/queue.h, [39](#)
 - include/public/stack.h, [40](#)
 - include/public/vector.h, [42](#)
- I
 - gdt_generic_datatype, [28](#)
- list, [29](#)
- list_append
 - Public interface to generic list data structure, [13](#)
- list_create
 - Public interface to generic list data structure, [14](#)
- list_delete_back
 - Public interface to generic list data structure, [14](#)
- list_delete_front
 - Public interface to generic list data structure, [14](#)
- list_delete_index
 - Public interface to generic list data structure, [14](#)
- list_destroy
 - Public interface to generic list data structure, [15](#)
- list_element_at_index
 - Public interface to generic list data structure, [15](#)
- list_find
 - Public interface to generic list data structure, [15](#)
- list_insert
 - Public interface to generic list data structure, [15](#)
- list_is_empty
 - Public interface to generic list data structure, [16](#)
- list_length
 - Public interface to generic list data structure, [16](#)
- list_node, [30](#)
- list_prepend
 - Public interface to generic list data structure, [16](#)
- list_set_element_at_index
 - Public interface to generic list data structure, [17](#)
- ll
 - gdt_generic_datatype, [28](#)
- p
 - gdt_generic_datatype, [28](#)
- pc
 - gdt_generic_datatype, [28](#)
- Private functionality for manipulating generic datatypes, [7](#)
 - DATATYPE_CHAR, [8](#)
 - DATATYPE_DOUBLE, [8](#)
 - DATATYPE_INT, [8](#)
 - DATATYPE_LONG, [8](#)
 - DATATYPE_LONG_LONG, [8](#)
 - DATATYPE_POINTER, [8](#)
 - DATATYPE_SIGNED_CHAR, [8](#)
 - DATATYPE_SIZE_T, [8](#)
 - DATATYPE_STRING, [8](#)
 - DATATYPE_UNSIGNED_CHAR, [8](#)
 - DATATYPE_UNSIGNED_INT, [8](#)
 - DATATYPE_UNSIGNED_LONG, [8](#)
 - DATATYPE_UNSIGNED_LONG_LONG, [8](#)
 - gds_datatype, [8](#)
 - gdt_compare, [8](#)
 - gdt_compare_void, [8](#)
 - gdt_free, [9](#)
 - gdt_get_value, [9](#)
 - gdt_reverse_compare_void, [9](#)
 - gdt_set_value, [9](#)
- Public general generic data structures functionality, [11](#)
 - GDS_EXIT_ON_ERROR, [11](#)
 - GDS_FREE_ON_DESTROY, [11](#)
 - GDS_RESIZABLE, [11](#)
 - gds_assert_quit, [11](#)
 - gds_error_quit, [11](#)
 - gds_option, [11](#)
 - gds_strerror_quit, [12](#)
- Public interface to generic list data structure, [13](#)
 - list_append, [13](#)
 - list_create, [14](#)
 - list_delete_back, [14](#)
 - list_delete_front, [14](#)
 - list_delete_index, [14](#)
 - list_destroy, [15](#)
 - list_element_at_index, [15](#)
 - list_find, [15](#)
 - list_insert, [15](#)
 - list_is_empty, [16](#)
 - list_length, [16](#)
 - list_prepend, [16](#)
 - list_set_element_at_index, [17](#)
- Public interface to generic queue data structure, [18](#)
 - queue_capacity, [18](#)
 - queue_create, [18](#)
 - queue_destroy, [19](#)
 - queue_free_space, [19](#)
 - queue_is_empty, [19](#)
 - queue_is_full, [19](#)
 - queue_peek, [20](#)
 - queue_pop, [20](#)
 - queue_push, [20](#)
 - queue_size, [21](#)
- Public interface to generic stack data structure, [22](#)
 - stack_capacity, [22](#)
 - stack_create, [22](#)

- stack_destroy, [23](#)
 - stack_free_space, [23](#)
 - stack_is_empty, [23](#)
 - stack_is_full, [23](#)
 - stack_peek, [24](#)
 - stack_pop, [24](#)
 - stack_push, [24](#)
 - stack_size, [25](#)
- Public interface to generic vector data structure., [26](#)
- queue, [30](#)
- queue_capacity
 - Public interface to generic queue data structure, [18](#)
- queue_create
 - Public interface to generic queue data structure, [18](#)
- queue_destroy
 - Public interface to generic queue data structure, [19](#)
- queue_free_space
 - Public interface to generic queue data structure, [19](#)
- queue_is_empty
 - Public interface to generic queue data structure, [19](#)
- queue_is_full
 - Public interface to generic queue data structure, [19](#)
- queue_peek
 - Public interface to generic queue data structure, [20](#)
- queue_pop
 - Public interface to generic queue data structure, [20](#)
- queue_push
 - Public interface to generic queue data structure, [20](#)
- queue_size
 - Public interface to generic queue data structure, [21](#)
- sc
 - gdt_generic_datatype, [28](#)
- st
 - gdt_generic_datatype, [28](#)
- stack, [31](#)
- stack_capacity
 - Public interface to generic stack data structure, [22](#)
- stack_create
 - Public interface to generic stack data structure, [22](#)
- stack_destroy
 - Public interface to generic stack data structure, [23](#)
- stack_free_space
 - Public interface to generic stack data structure, [23](#)
- stack_is_empty
 - Public interface to generic stack data structure, [23](#)
- stack_is_full
 - Public interface to generic stack data structure, [23](#)
- stack_peek
 - Public interface to generic stack data structure, [24](#)
- stack_pop
 - Public interface to generic stack data structure, [24](#)
- stack_push
 - Public interface to generic stack data structure, [24](#)
- stack_size
 - Public interface to generic stack data structure, [25](#)
- type
 - gdt_generic_datatype, [28](#)
 - uc
 - gdt_generic_datatype, [28](#)
 - ui
 - gdt_generic_datatype, [28](#)
 - ul
 - gdt_generic_datatype, [29](#)
 - ull
 - gdt_generic_datatype, [29](#)
 - vector, [32](#)