

gds

Generated by Doxygen 1.8.1.2

Sat Nov 8 2014 22:07:03

Contents

1	Generic Data Structures Library	1
2	Todo List	3
3	Module Index	5
3.1	Modules	5
4	Data Structure Index	7
4.1	Data Structures	7
5	File Index	9
5.1	File List	9
6	Module Documentation	11
6.1	Private functionality for manipulating generic datatypes	11
6.1.1	Detailed Description	12
6.1.2	Typedef Documentation	12
6.1.2.1	gds_cfunc	12
6.1.3	Enumeration Type Documentation	12
6.1.3.1	gds_datatype	12
6.1.4	Function Documentation	12
6.1.4.1	gdt_compare	12
6.1.4.2	gdt_compare_void	13
6.1.4.3	gdt_free	13
6.1.4.4	gdt_get_value	13
6.1.4.5	gdt_reverse_compare_void	13
6.1.4.6	gdt_set_value	14
6.2	Public general generic data structures functionality	15
6.2.1	Detailed Description	15
6.2.2	Enumeration Type Documentation	15
6.2.2.1	gds_option	15
6.2.3	Function Documentation	15
6.2.3.1	gds_assert_quit	15

6.2.3.2	<code>gds_error_quit</code>	15
6.2.3.3	<code>gds_strerror_quit</code>	16
6.3	Public interface to generic list data structure	17
6.3.1	Detailed Description	17
6.3.2	Typedef Documentation	17
6.3.2.1	List	17
6.3.3	Function Documentation	18
6.3.3.1	<code>list_append</code>	18
6.3.3.2	<code>list_create</code>	18
6.3.3.3	<code>list_delete_back</code>	18
6.3.3.4	<code>list_delete_front</code>	18
6.3.3.5	<code>list_delete_index</code>	19
6.3.3.6	<code>list_destroy</code>	19
6.3.3.7	<code>list_element_at_index</code>	19
6.3.3.8	<code>list_find</code>	19
6.3.3.9	<code>list_insert</code>	20
6.3.3.10	<code>list_is_empty</code>	20
6.3.3.11	<code>list_length</code>	20
6.3.3.12	<code>list_prepend</code>	21
6.3.3.13	<code>list_set_element_at_index</code>	21
6.4	Public interface to generic queue data structure	22
6.4.1	Detailed Description	22
6.4.2	Typedef Documentation	22
6.4.2.1	Queue	22
6.4.3	Function Documentation	22
6.4.3.1	<code>queue_capacity</code>	22
6.4.3.2	<code>queue_create</code>	23
6.4.3.3	<code>queue_destroy</code>	23
6.4.3.4	<code>queue_free_space</code>	23
6.4.3.5	<code>queue_is_empty</code>	23
6.4.3.6	<code>queue_is_full</code>	24
6.4.3.7	<code>queue_peek</code>	24
6.4.3.8	<code>queue_pop</code>	24
6.4.3.9	<code>queue_push</code>	25
6.4.3.10	<code>queue_size</code>	25
6.5	Public interface to generic stack data structure	26
6.5.1	Detailed Description	26
6.5.2	Typedef Documentation	26
6.5.2.1	Stack	26
6.5.3	Function Documentation	26

6.5.3.1	stack_capacity	26
6.5.3.2	stack_create	27
6.5.3.3	stack_destroy	27
6.5.3.4	stack_free_space	27
6.5.3.5	stack_is_empty	27
6.5.3.6	stack_is_full	28
6.5.3.7	stack_peek	28
6.5.3.8	stack_pop	28
6.5.3.9	stack_push	29
6.5.3.10	stack_size	29
6.6	Public interface to generic vector data structure.	30
6.6.1	Detailed Description	30
6.6.2	Typedef Documentation	31
6.6.2.1	Vector	31
6.6.3	Function Documentation	31
6.6.3.1	vector_append	31
6.6.3.2	vector_capacity	31
6.6.3.3	vector_create	31
6.6.3.4	vector_delete_back	32
6.6.3.5	vector_delete_front	32
6.6.3.6	vector_delete_index	32
6.6.3.7	vector_destroy	32
6.6.3.8	vector_element_at_index	33
6.6.3.9	vector_find	33
6.6.3.10	vector_free_space	33
6.6.3.11	vector_insert	34
6.6.3.12	vector_is_empty	34
6.6.3.13	vector_length	34
6.6.3.14	vector_prepend	34
6.6.3.15	vector_reverse_sort	35
6.6.3.16	vector_set_element_at_index	35
6.6.3.17	vector_sort	35
7	Data Structure Documentation	37
7.1	gdt_generic_datatype Struct Reference	37
7.1.1	Detailed Description	37
7.1.2	Field Documentation	37
7.1.2.1	c	37
7.1.2.2	compfunc	38
7.1.2.3	d	38

7.1.2.4	data	38
7.1.2.5	i	38
7.1.2.6	l	38
7.1.2.7	ll	38
7.1.2.8	p	38
7.1.2.9	pc	38
7.1.2.10	sc	38
7.1.2.11	st	38
7.1.2.12	type	38
7.1.2.13	uc	38
7.1.2.14	ui	39
7.1.2.15	ul	39
7.1.2.16	ull	39
7.2	list Struct Reference	39
7.2.1	Detailed Description	40
7.2.2	Field Documentation	40
7.2.2.1	compfunc	40
7.2.2.2	exit_on_error	40
7.2.2.3	free_on_destroy	40
7.2.2.4	head	40
7.2.2.5	length	40
7.2.2.6	tail	40
7.2.2.7	type	40
7.3	list_node Struct Reference	41
7.3.1	Detailed Description	41
7.3.2	Field Documentation	41
7.3.2.1	element	41
7.3.2.2	next	41
7.3.2.3	prev	41
7.4	queue Struct Reference	42
7.4.1	Detailed Description	42
7.4.2	Field Documentation	42
7.4.2.1	back	42
7.4.2.2	capacity	42
7.4.2.3	elements	42
7.4.2.4	exit_on_error	42
7.4.2.5	free_on_destroy	43
7.4.2.6	front	43
7.4.2.7	resizable	43
7.4.2.8	size	43

7.4.2.9	type	43
7.5	stack Struct Reference	43
7.5.1	Detailed Description	44
7.5.2	Field Documentation	44
7.5.2.1	capacity	44
7.5.2.2	elements	44
7.5.2.3	exit_on_error	44
7.5.2.4	free_on_destroy	44
7.5.2.5	resizable	44
7.5.2.6	top	44
7.5.2.7	type	44
7.6	vector Struct Reference	45
7.6.1	Detailed Description	45
7.6.2	Field Documentation	45
7.6.2.1	capacity	45
7.6.2.2	compfunc	45
7.6.2.3	elements	45
7.6.2.4	exit_on_error	45
7.6.2.5	free_on_destroy	46
7.6.2.6	length	46
7.6.2.7	type	46
8	File Documentation	47
8.1	gds.dox File Reference	47
8.2	include/private/gds_common.h File Reference	47
8.2.1	Detailed Description	48
8.3	include/private/gdt.dox File Reference	48
8.4	include/private/gdt.h File Reference	48
8.4.1	Detailed Description	50
8.5	include/public/gds_public_types.h File Reference	50
8.5.1	Detailed Description	51
8.6	include/public/gds_util.h File Reference	51
8.6.1	Detailed Description	52
8.7	include/public/general.dox File Reference	52
8.8	include/public/list.dox File Reference	52
8.9	include/public/list.h File Reference	52
8.9.1	Detailed Description	54
8.10	include/public/queue.dox File Reference	54
8.11	include/public/queue.h File Reference	54
8.11.1	Detailed Description	56

8.12	include/public/stack.dox File Reference	56
8.13	include/public/stack.h File Reference	56
8.13.1	Detailed Description	58
8.14	include/public/vector.dox File Reference	58
8.15	include/public/vector.h File Reference	58
8.15.1	Detailed Description	60
8.16	src/gds_util.c File Reference	60
8.16.1	Detailed Description	61
8.17	src/gdt.c File Reference	61
8.17.1	Detailed Description	63
8.17.2	Function Documentation	63
8.17.2.1	gdt_compare_char	63
8.17.2.2	gdt_compare_double	63
8.17.2.3	gdt_compare_int	64
8.17.2.4	gdt_compare_long	64
8.17.2.5	gdt_compare_longlong	64
8.17.2.6	gdt_compare_schar	64
8.17.2.7	gdt_compare_sizet	65
8.17.2.8	gdt_compare_string	65
8.17.2.9	gdt_compare_uchar	65
8.17.2.10	gdt_compare_uint	66
8.17.2.11	gdt_compare_ulong	66
8.17.2.12	gdt_compare_ulonglong	66
8.18	src/list.c File Reference	66
8.18.1	Detailed Description	68
8.18.2	Typedef Documentation	68
8.18.2.1	ListNode	68
8.18.3	Function Documentation	68
8.18.3.1	list_insert_internal	68
8.18.3.2	list_node_at_index	69
8.18.3.3	list_node_create	69
8.18.3.4	list_node_destroy	69
8.19	src/queue.c File Reference	69
8.19.1	Detailed Description	71
8.19.2	Variable Documentation	71
8.19.2.1	GROWTH	71
8.20	src/stack.c File Reference	71
8.20.1	Detailed Description	73
8.20.2	Variable Documentation	73
8.20.2.1	GROWTH	73

8.21	src/vector.c File Reference	73
8.21.1	Detailed Description	75
8.21.2	Function Documentation	75
8.21.2.1	vector_insert_internal	75
8.21.3	Variable Documentation	75
8.21.3.1	GROWTH	75

Chapter 1

Generic Data Structures Library

GDS is a C language generic data structures library.

Chapter 2

Todo List

File `list.c`

Implement sorting.

Implement iterators.

Global `queue_push` (Queue queue,...)

Rewrite to move only the required elements

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

Private functionality for manipulating generic datatypes	11
Public general generic data structures functionality	15
Public interface to generic list data structure	17
Public interface to generic queue data structure	22
Public interface to generic stack data structure	26
Public interface to generic vector data structure.	30

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

gdt_generic_datatype	
Generic datatype structure	37
list	39
list_node	41
queue	42
stack	43
vector	45

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

include/private/ gds_common.h	
Common internal headers for data structures	47
include/private/ gdt.h	
Interface to generic data element functionality	48
include/public/ gds_public_types.h	
Common public types for generic data structures library	50
include/public/ gds_util.h	
Interface to general utility functions	51
include/public/ list.h	
Interface to generic list data structure	52
include/public/ queue.h	
Interface to generic queue data structure	54
include/public/ stack.h	
Interface to generic stack data structure	56
include/public/ vector.h	
Interface to generic vector data structure	58
src/ gds_util.c	
Implementation of general utility functions	60
src/ gdt.c	
Implementation of generic data element functionality	61
src/ list.c	
Implementation of generic list data structure	66
src/ queue.c	
Implementation of generic queue data structure	69
src/ stack.c	
Implementation of generic stack data structure	71
src/ vector.c	
Implementation of generic vector data structure	73

Chapter 6

Module Documentation

6.1 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.

6.1.1 Detailed Description

This module implements the mechanism for allowing generic datatypes. Each datatype implements a C `union` containing all the allowable fundamental types. Functions are provided for getting, setting, `free()`ing, and comparing values.

6.1.2 Typedef Documentation

6.1.2.1 `typedef int(* gds_cfunc)(const void *, const void *)`

Type definition for comparison function pointer.

6.1.3 Enumeration Type Documentation

6.1.3.1 `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 *

6.1.4 Function Documentation

6.1.4.1 `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

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

6.1.4.2 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>p1</i>	A pointer to the first generic datatype.
<i>p2</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.

6.1.4.3 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.
-------------	------------------------------------

6.1.4.4 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.

6.1.4.5 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>p1</i>	A pointer to the first generic datatype.
<i>p2</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.

6.1.4.6 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 DATATYPE_POINTER. For DATATYPE_POINTER, 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.

6.2 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().

6.2.1 Detailed Description

This module contains general functionality used with or by the other data structures, including common creation options, and functions for outputting error messages.

6.2.2 Enumeration Type Documentation

6.2.2.1 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

6.2.3 Function Documentation

6.2.3.1 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

<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.

6.2.3.2 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.

6.2.3.3 `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.

6.3 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.

6.3.1 Detailed Description

A list is data structure containing a finite ordered collection of values which allows sequential access (compared to a vector, or array, which allows random access).

6.3.2 Typedef Documentation

6.3.2.1 typedef struct list* List

Opaque list type definition.

6.3.3 Function Documentation

6.3.3.1 `bool list_append (List list, ...)`

Appends a value to the back of a list.

Parameters

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

6.3.3.2 `List list_create (const enum gds_datatype type, const int opts, ...)`

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.

6.3.3.3 `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.

6.3.3.4 `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.

6.3.3.5 `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.

6.3.3.6 `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.
-------------	------------------------

6.3.3.7 `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.

6.3.3.8 `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.
...	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

6.3.3.9 `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.
...	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.

6.3.3.10 `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

6.3.3.11 `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.

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

Prepends a value to the front of a list.

Parameters

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

6.3.3.13 `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.

6.4 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.

6.4.1 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.

6.4.2 Typedef Documentation

6.4.2.1 typedef struct queue* Queue

Opaque queue type definition.

6.4.3 Function Documentation

6.4.3.1 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

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

Returns

The capacity of the queue.

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

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_RESIZABLE to dynamically resize the queue on-demand; 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.

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

6.4.3.4 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.

6.4.3.5 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

6.4.3.6 `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

6.4.3.7 `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>queue</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.

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

Pops a value from the queue.

Parameters

<i>queue</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.

6.4.3.9 `bool queue.push (Queue queue, ...)`

Pushes a value onto the queue.

Parameters

<i>queue</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.

Todo Rewrite to move only the required elements

6.4.3.10 `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.

6.5 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.

6.5.1 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.

6.5.2 Typedef Documentation

6.5.2.1 typedef struct `stack`* `Stack`

Opaque stack type definition.

6.5.3 Function Documentation

6.5.3.1 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

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

Returns

The capacity of the stack.

6.5.3.2 Stack `stack_create (const size_t capacity, const enum gds_datatype type, const int opts)`

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_RESIZABLE</code> to dynamically resize the stack on-demand; <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.

6.5.3.3 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.
--------------	-------------------------

6.5.3.4 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.

6.5.3.5 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

6.5.3.6 `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

6.5.3.7 `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>stack</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.

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

Pops a value from the stack.

Parameters

<i>stack</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.

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

Pushes a value onto the stack.

Parameters

<i>stack</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.

6.5.3.10 `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.

6.6 Public interface to generic vector data structure.

Typedefs

- typedef struct `vector` * `Vector`
Opaque vector type definition.

Functions

- `Vector vector_create` (const size_t capacity, const enum `gds_datatype` type, const int opts,...)
Creates a new vector.
- void `vector_destroy` (`Vector vector`)
Destroys a vector.
- bool `vector_append` (`Vector vector`,...)
Appends a value to the back of a vector.
- bool `vector_prepend` (`Vector vector`,...)
Prepends a value to the front of a vector.
- bool `vector_insert` (`Vector vector`, const size_t index,...)
Inserts a value into a vector.
- bool `vector_delete_front` (`Vector vector`)
Deletes the value at the front of the vector.
- bool `vector_delete_back` (`Vector vector`)
Deletes the value at the back of the vector.
- bool `vector_delete_index` (`Vector vector`, const size_t index)
Deletes the value at the specified index of the vector.
- bool `vector_element_at_index` (`Vector vector`, const size_t index, void *p)
Gets the value at the specified index of the vector.
- bool `vector_set_element_at_index` (`Vector vector`, const size_t index,...)
Sets the value at the specified index of the vector.
- bool `vector_find` (`Vector vector`, size_t *index,...)
Tests if a value is contained in a vector.
- void `vector_sort` (`Vector vector`)
Sorts a vector in-place, in ascending order.
- void `vector_reverse_sort` (`Vector vector`)
Sorts a vector in-place, in descending order.
- bool `vector_is_empty` (`Vector vector`)
Tests if a vector is empty.
- size_t `vector_length` (`Vector vector`)
Returns the length of a vector.
- size_t `vector_capacity` (`Vector vector`)
Returns the capacity of a vector.
- size_t `vector_free_space` (`Vector vector`)
Returns the free space in a vector.

6.6.1 Detailed Description

A vector (or array) is a data structure containing a finite ordered collection of values which allows random access (compared to a list, which only allows sequential access).

6.6.2 Typedef Documentation

6.6.2.1 typedef struct vector* Vector

Opaque vector type definition.

6.6.3 Function Documentation

6.6.3.1 bool vector_append (Vector vector, ...)

Appends a value to the back of a vector.

Parameters

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

Return values

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

6.6.3.2 size_t vector_capacity (Vector vector)

Returns the capacity of a vector.

The capacity of the vector is equivalent to the number of values it is capable of holding. This value can dynamically change if a vector resizes to append an element at the back of the vector. The capacity does not change when elements are deleted from a vector.

Parameters

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

Returns

The capacity of the vector.

6.6.3.3 Vector vector_create (const size_t capacity, const enum gds_datatype type, const int opts, ...)

Creates a new vector.

Parameters

<i>capacity</i>	The initial capacity for the vector.
<i>type</i>	The datatype for the vector.
<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 vector is destroyed. If this option is specified, then the caller should ensure that all the elements of the vector have been initialized prior to destruction.
- `GDS_EXIT_ON_ERROR` to print a message to the standard error stream and `exit()`, rather than returning a failure status.

Parameters

...	If <code>type</code> 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>	Vector creation failed.
<i>non-NULL</i>	A pointer to the new vector.

6.6.3.4 `bool vector_delete_back (Vector vector)`

Deletes the value at the back of the vector.

Parameters

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

Return values

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

6.6.3.5 `bool vector_delete_front (Vector vector)`

Deletes the value at the front of the vector.

Parameters

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

Return values

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

6.6.3.6 `bool vector_delete_index (Vector vector, const size_t index)`

Deletes the value at the specified index of the vector.

Parameters

<i>vector</i>	A pointer to the vector.
<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.

6.6.3.7 `void vector_destroy (Vector vector)`

Destroys a vector.

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

Parameters

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

6.6.3.8 bool vector_element_at_index (Vector vector, const size_t index, void * p)

Gets the value at the specified index of the vector.

Parameters

<i>vector</i>	A pointer to the vector.
<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 vector. 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.

6.6.3.9 bool vector_find (Vector vector, size_t * index, ...)

Tests if a value is contained in a vector.

Parameters

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

Return values

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

6.6.3.10 size_t vector_free_space (Vector vector)

Returns the free space in a vector.

The free space in a vector is equivalent to its capacity less its length. The free space can change if a vector dynamically resizes to append an element at the back of the vector, or if elements are deleted from the vector.

Parameters

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

Returns

The free space in the vector.

6.6.3.11 `bool vector_insert (Vector vector, const size_t index, ...)`

Inserts a value into a vector.

Parameters

<i>vector</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 vector. This should be of a type appropriate to the type set when creating the vector.

Return values

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

6.6.3.12 `bool vector_is_empty (Vector vector)`

Tests if a vector is empty.

Parameters

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

Return values

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

6.6.3.13 `size_t vector_length (Vector vector)`

Returns the length of a vector.

The length of the vector is equivalent to the number of values it contains. This can be less than the initial capacity, and as low as zero, if elements have been deleted from the vector.

Parameters

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

Returns

The length of the vector.

6.6.3.14 `bool vector_prepend (Vector vector, ...)`

Prepends a value to the front of a vector.

Parameters

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

Return values

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

6.6.3.15 void vector_reverse_sort (Vector vector)

Sorts a vector in-place, in descending order.

Parameters

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

6.6.3.16 bool vector_set_element_at_index (Vector vector, const size_t index, ...)

Sets the value at the specified index of the vector.

Parameters

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

Return values

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

6.6.3.17 void vector_sort (Vector vector)

Sorts a vector in-place, in ascending order.

Parameters

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

Chapter 7

Data Structure Documentation

7.1 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](#)

7.1.1 Detailed Description

Generic datatype structure.

7.1.2 Field Documentation

7.1.2.1 char gdt_generic_datatype::c

char

7.1.2.2 gds_cfunc gdt_generic_datatype::compfunc

Comparison function pointer

7.1.2.3 double gdt_generic_datatype::d

double

7.1.2.4 union { ... } gdt_generic_datatype::data

Data union

7.1.2.5 int gdt_generic_datatype::i

int

7.1.2.6 long gdt_generic_datatype::l

long

7.1.2.7 long long int gdt_generic_datatype::ll

long long

7.1.2.8 void* gdt_generic_datatype::p

void *

7.1.2.9 char* gdt_generic_datatype::pc

char *, string

7.1.2.10 signed char gdt_generic_datatype::sc

signed char

7.1.2.11 size_t gdt_generic_datatype::st

size_t

7.1.2.12 enum gds_datatype gdt_generic_datatype::type

Data type

7.1.2.13 unsigned char gdt_generic_datatype::uc

unsigned char

7.1.2.14 unsigned int gdt_generic_datatype::ui

unsigned int

7.1.2.15 unsigned long gdt_generic_datatype::ul

unsigned long

7.1.2.16 unsigned long long int gdt_generic_datatype::ull

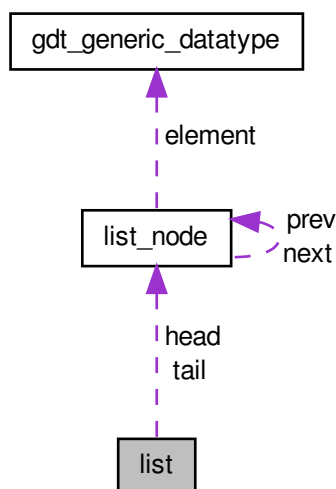
unsigned long long

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

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

7.2 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](#)

7.2.1 Detailed Description

List structure

7.2.2 Field Documentation

7.2.2.1 `gds_cfunc list::compfunc`

Element comparison function

7.2.2.2 `bool list::exit_on_error`

Exit on error if true

7.2.2.3 `bool list::free_on_destroy`

Free pointer elements on destroy if true

7.2.2.4 `struct list_node* list::head`

Pointer to head of list

7.2.2.5 `size_t list::length`

Length of list

7.2.2.6 `struct list_node* list::tail`

Pointer to tail of list

7.2.2.7 `enum gds_datatype list::type`

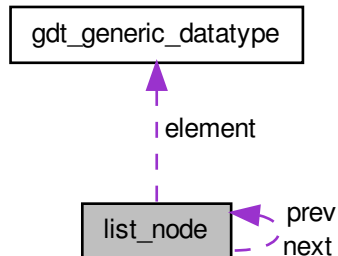
List datatype

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

- [src/list.c](#)

7.3 list_node Struct Reference

Collaboration diagram for list_node:



Data Fields

- struct [gdt_generic_datatype](#) `element`
- struct [list_node](#) * `prev`
- struct [list_node](#) * `next`

7.3.1 Detailed Description

List node structure

7.3.2 Field Documentation

7.3.2.1 struct `gdt_generic_datatype` `list_node::element`

Data element

7.3.2.2 struct `list_node`* `list_node::next`

Pointer to next node

7.3.2.3 struct `list_node`* `list_node::prev`

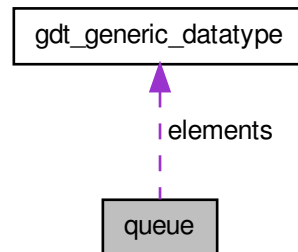
Pointer to previous node

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

- [src/list.c](#)

7.4 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](#)

7.4.1 Detailed Description

Queue structure

7.4.2 Field Documentation

7.4.2.1 size_t queue::back

Back of queue

7.4.2.2 size_t queue::capacity

Capacity of queue

7.4.2.3 struct gdt_generic_datatype* queue::elements

Pointer to elements

7.4.2.4 bool queue::exit_on_error

Exit on error if true

7.4.2.5 bool queue::free_on_destroy

Free pointer elements on destroy if true

7.4.2.6 size_t queue::front

Front of queue

7.4.2.7 bool queue::resizable

Dynamically resizable if true

7.4.2.8 size_t queue::size

Size of queue

7.4.2.9 enum gds_datatype queue::type

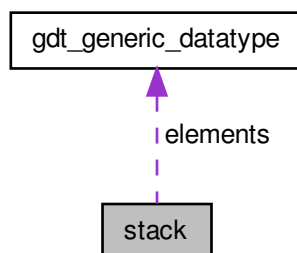
Queue datatype

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

- [src/queue.c](#)

7.5 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](#)

7.5.1 Detailed Description

Stack structure

7.5.2 Field Documentation

7.5.2.1 `size_t stack::capacity`

Stack capacity

7.5.2.2 `struct gdt_generic_datatype* stack::elements`

Pointer to elements

7.5.2.3 `bool stack::exit_on_error`

Exit on error if true

7.5.2.4 `bool stack::free_on_destroy`

Free pointer elements on destroy if true

7.5.2.5 `bool stack::resizable`

Dynamically resizable if true

7.5.2.6 `size_t stack::top`

Top of stack

7.5.2.7 `enum gds_datatype stack::type`

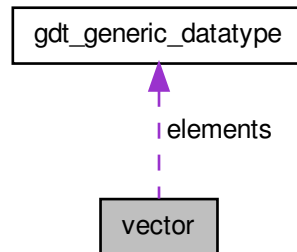
Stack datatype

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

- [src/stack.c](#)

7.6 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](#)

7.6.1 Detailed Description

Vector structure

7.6.2 Field Documentation

7.6.2.1 `size_t` `vector::capacity`

Vector capacity

7.6.2.2 `int`(* `vector::compfunc`)(const void *, const void *)

Compare function

7.6.2.3 `struct` [gdt_generic_datatype](#)* `vector::elements`

Pointer to elements

7.6.2.4 `bool` `vector::exit_on_error`

Exit on error if true

7.6.2.5 `bool vector::free_on_destroy`

Free pointer elements on destroy if true

7.6.2.6 `size_t vector::length`

Vector length

7.6.2.7 `enum gds_datatype vector::type`

Vector datatype

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

- [src/vector.c](#)

Chapter 8

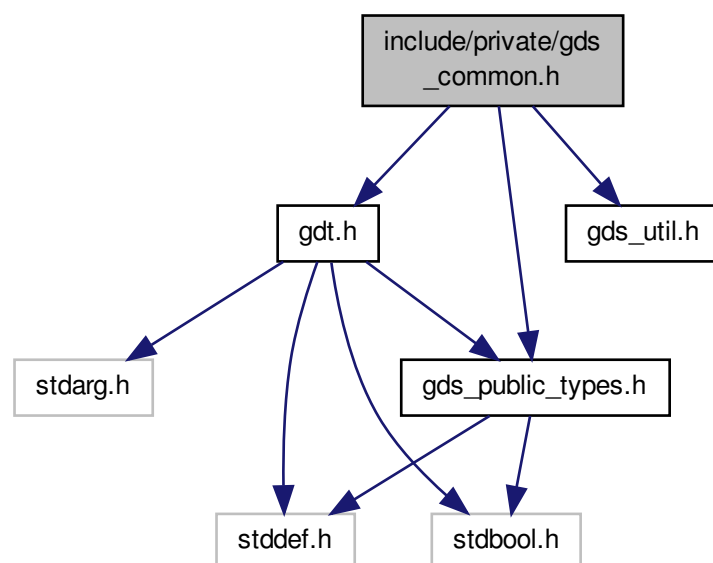
File Documentation

8.1 gds.dox File Reference

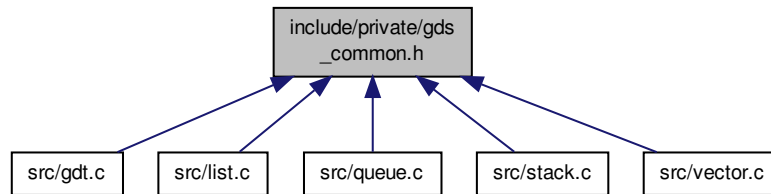
8.2 include/private/gds_common.h File Reference

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:
```



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



8.2.1 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/>

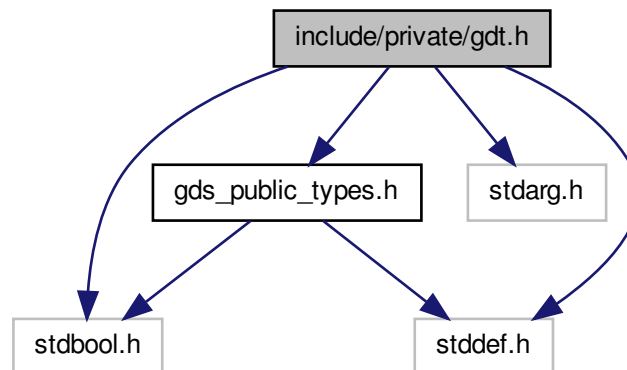
8.3 include/private/gdt.dox File Reference

8.4 include/private/gdt.h File Reference

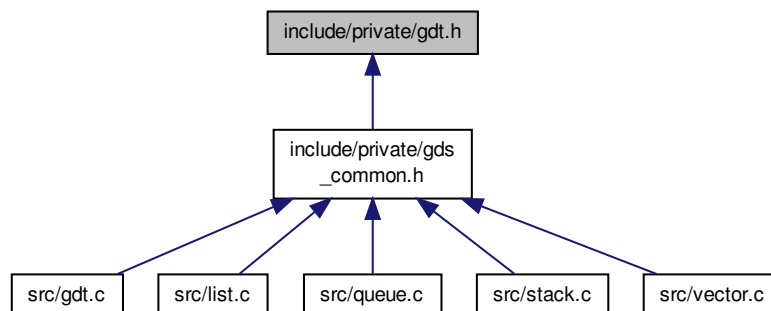
Interface to generic data element functionality.

```
#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.

8.4.1 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/>

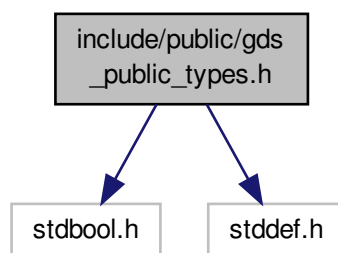
8.5 include/public/gds_public_types.h File Reference

Common public types for generic data structures library.

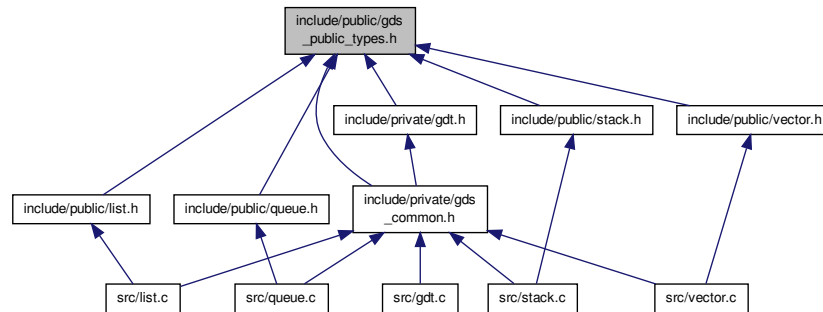
```
#include <stdbool.h>
```

```
#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.

8.5.1 Detailed Description

Common public types for generic data structures library.

Author

Paul Griffiths

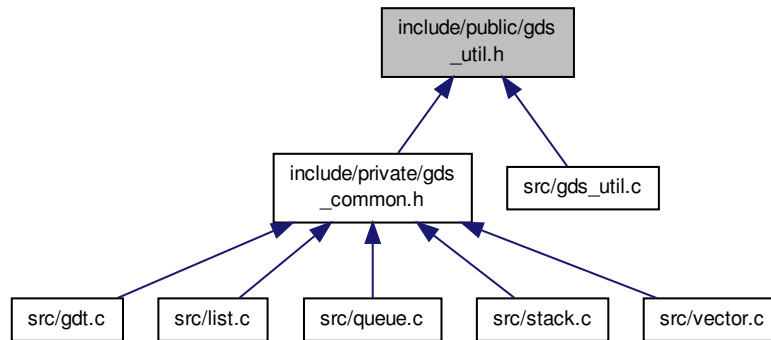
Copyright

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

8.6 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().

8.6.1 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/>

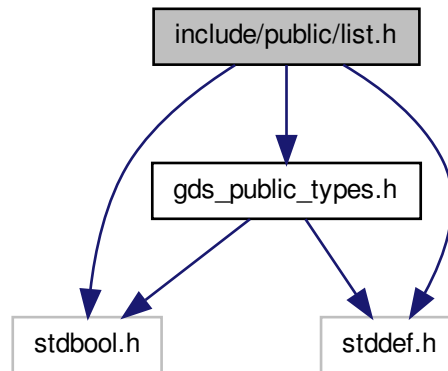
8.7 include/public/general.dox File Reference

8.8 include/public/list.dox File Reference

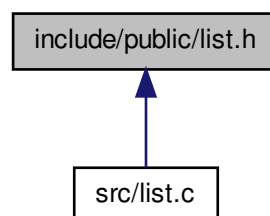
8.9 include/public/list.h File Reference

Interface to generic list data structure.

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



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



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.

8.9.1 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/>

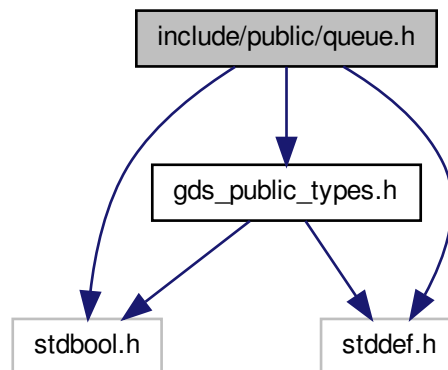
8.10 include/public/queue.dox File Reference

8.11 include/public/queue.h File Reference

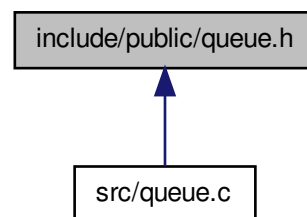
Interface to generic queue data structure.

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


Include dependency graph for queue.h:



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



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.

8.11.1 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/>

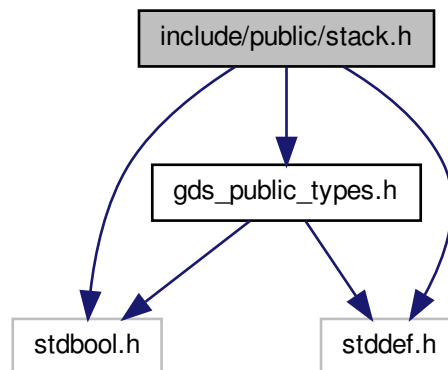
8.12 include/public/stack.dox File Reference

8.13 include/public/stack.h File Reference

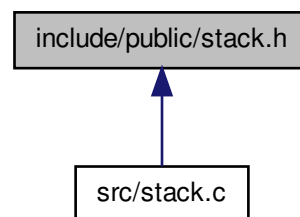
Interface to generic stack data structure.

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

Include dependency graph for stack.h:



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



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.

8.13.1 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/>

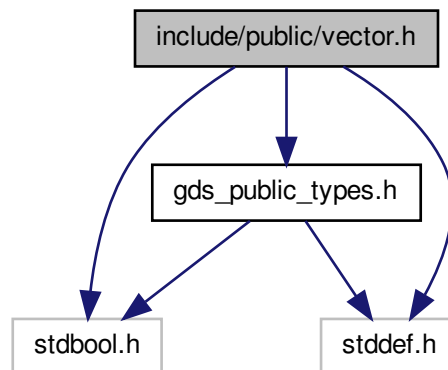
8.14 `include/public/vector.dox` File Reference

8.15 `include/public/vector.h` File Reference

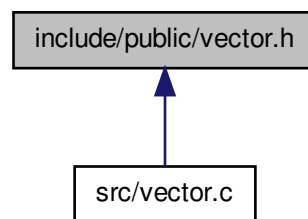
Interface to generic vector data structure.

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

Include dependency graph for vector.h:



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



Typedefs

- typedef struct `vector` * `Vector`
Opaque vector type definition.

Functions

- `Vector vector_create` (const size_t capacity, const enum `gds_datatype` type, const int opts,...)
Creates a new vector.
- void `vector_destroy` (`Vector vector`)
Destroys a vector.
- bool `vector_append` (`Vector vector`,...)
Appends a value to the back of a vector.
- bool `vector_prepend` (`Vector vector`,...)
Prepends a value to the front of a vector.

- bool `vector_insert` (`Vector vector`, const size_t index,...)
Inserts a value into a vector.
- bool `vector_delete_front` (`Vector vector`)
Deletes the value at the front of the vector.
- bool `vector_delete_back` (`Vector vector`)
Deletes the value at the back of the vector.
- bool `vector_delete_index` (`Vector vector`, const size_t index)
Deletes the value at the specified index of the vector.
- bool `vector_element_at_index` (`Vector vector`, const size_t index, void *p)
Gets the value at the specified index of the vector.
- bool `vector_set_element_at_index` (`Vector vector`, const size_t index,...)
Sets the value at the specified index of the vector.
- bool `vector_find` (`Vector vector`, size_t *index,...)
Tests if a value is contained in a vector.
- void `vector_sort` (`Vector vector`)
Sorts a vector in-place, in ascending order.
- void `vector_reverse_sort` (`Vector vector`)
Sorts a vector in-place, in descending order.
- bool `vector_is_empty` (`Vector vector`)
Tests if a vector is empty.
- size_t `vector_length` (`Vector vector`)
Returns the length of a vector.
- size_t `vector_capacity` (`Vector vector`)
Returns the capacity of a vector.
- size_t `vector_free_space` (`Vector vector`)
Returns the free space in a vector.

8.15.1 Detailed Description

Interface to generic vector data structure.

Author

Paul Griffiths

Copyright

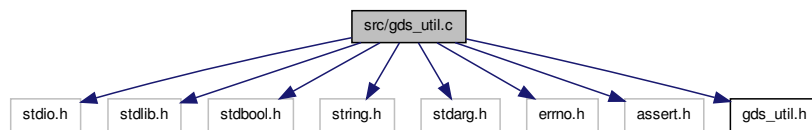
Copyright 2014 Paul Griffiths. Distributed under the terms of the GNU General Public License. <http://www.gnu.org/licenses/>

8.16 `src/gds_util.c` File Reference

Implementation of general utility functions.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <stdarg.h>
#include <errno.h>
#include <assert.h>
#include "gds_util.h"
```

Include dependency graph for gds_util.c:



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().

8.16.1 Detailed Description

Implementation of 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/>

8.17 src/gdt.c File Reference

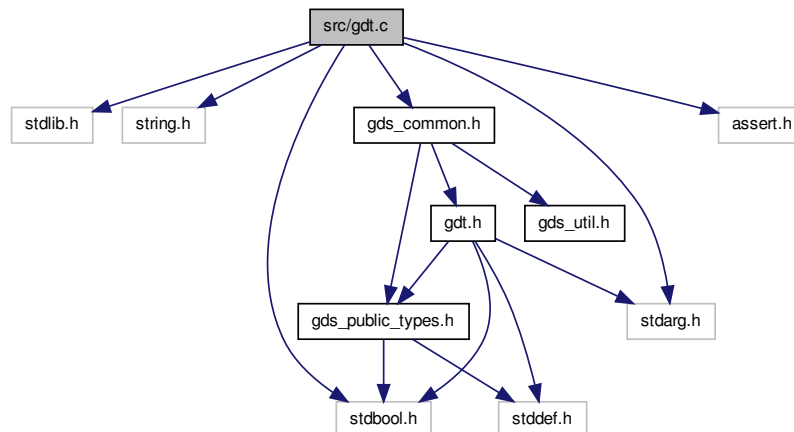
Implementation of generic data element functionality.

```

#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <assert.h>
#include <stdarg.h>
#include "gds_common.h"

```

Include dependency graph for gdt.c:



Functions

- static int [gdt_compare_char](#) (const void *p1, const void *p2)
Compare function for char.
- static int [gdt_compare_uchar](#) (const void *p1, const void *p2)
Compare function for unsigned char.
- static int [gdt_compare_schar](#) (const void *p1, const void *p2)
Compare function for signed char.
- static int [gdt_compare_int](#) (const void *p1, const void *p2)
Compare function for int.
- static int [gdt_compare_uint](#) (const void *p1, const void *p2)
Compare function for unsigned int.
- static int [gdt_compare_long](#) (const void *p1, const void *p2)
Compare function for long.
- static int [gdt_compare_ulong](#) (const void *p1, const void *p2)
Compare function for unsigned long.
- static int [gdt_compare_longlong](#) (const void *p1, const void *p2)
Compare function for long long.
- static int [gdt_compare_ulonglong](#) (const void *p1, const void *p2)
Compare function for unsigned long long.
- static int [gdt_compare_sizet](#) (const void *p1, const void *p2)
Compare function for size_t.
- static int [gdt_compare_double](#) (const void *p1, const void *p2)
Compare function for double.
- static int [gdt_compare_string](#) (const void *p1, const void *p2)
Compare function for string.
- 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.

8.17.1 Detailed Description

Implementation of 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/>

8.17.2 Function Documentation

8.17.2.1 static int `gdt_compare_char` (const void * *p1*, const void * *p2*) [static]

Compare function for char.

Parameters

<i>p1</i>	Pointer to first value
<i>p2</i>	Pointer to second value

Return values

0	First value is equal to second value
-1	First value is less than second value
1	First value is greater than second value

8.17.2.2 static int `gdt_compare_double` (const void * *p1*, const void * *p2*) [static]

Compare function for double.

Parameters

<i>p1</i>	Pointer to first value
<i>p2</i>	Pointer to second value

Return values

0	First value is equal to second value
-1	First value is less than second value
1	First value is greater than second value

8.17.2.3 static int gdt_compare_int (const void * *p1*, const void * *p2*) [static]

Compare function for int.

Parameters

<i>p1</i>	Pointer to first value
<i>p2</i>	Pointer to second value

Return values

0	First value is equal to second value
-1	First value is less than second value
1	First value is greater than second value

8.17.2.4 static int gdt_compare_long (const void * *p1*, const void * *p2*) [static]

Compare function for long.

Parameters

<i>p1</i>	Pointer to first value
<i>p2</i>	Pointer to second value

Return values

0	First value is equal to second value
-1	First value is less than second value
1	First value is greater than second value

8.17.2.5 static int gdt_compare_longlong (const void * *p1*, const void * *p2*) [static]

Compare function for long long.

Parameters

<i>p1</i>	Pointer to first value
<i>p2</i>	Pointer to second value

Return values

0	First value is equal to second value
-1	First value is less than second value
1	First value is greater than second value

8.17.2.6 static int gdt_compare_schar (const void * *p1*, const void * *p2*) [static]

Compare function for signed char.

Parameters

<i>p1</i>	Pointer to first value
<i>p2</i>	Pointer to second value

Return values

0	First value is equal to second value
-1	First value is less than second value
1	First value is greater than second value

8.17.2.7 static int gdt_compare_size (const void * *p1*, const void * *p2*) [static]

Compare function for size_t.

Parameters

<i>p1</i>	Pointer to first value
<i>p2</i>	Pointer to second value

Return values

0	First value is equal to second value
-1	First value is less than second value
1	First value is greater than second value

8.17.2.8 static int gdt_compare_string (const void * *p1*, const void * *p2*) [static]

Compare function for string.

Parameters

<i>p1</i>	Pointer to first value
<i>p2</i>	Pointer to second value

Return values

0	First value is equal to second value
-1	First value is less than second value
1	First value is greater than second value

8.17.2.9 static int gdt_compare_uchar (const void * *p1*, const void * *p2*) [static]

Compare function for unsigned char.

Parameters

<i>p1</i>	Pointer to first value
<i>p2</i>	Pointer to second value

Return values

0	First value is equal to second value
-1	First value is less than second value
1	First value is greater than second value

8.17.2.10 `static int gdt_compare_uint (const void * p1, const void * p2) [static]`

Compare function for unsigned int.

Parameters

<i>p1</i>	Pointer to first value
<i>p2</i>	Pointer to second value

Return values

0	First value is equal to second value
-1	First value is less than second value
1	First value is greater than second value

8.17.2.11 `static int gdt_compare_ulong (const void * p1, const void * p2) [static]`

Compare function for unsigned long.

Parameters

<i>p1</i>	Pointer to first value
<i>p2</i>	Pointer to second value

Return values

0	First value is equal to second value
-1	First value is less than second value
1	First value is greater than second value

8.17.2.12 `static int gdt_compare_ulonglong (const void * p1, const void * p2) [static]`

Compare function for unsigned long long.

Parameters

<i>p1</i>	Pointer to first value
<i>p2</i>	Pointer to second value

Return values

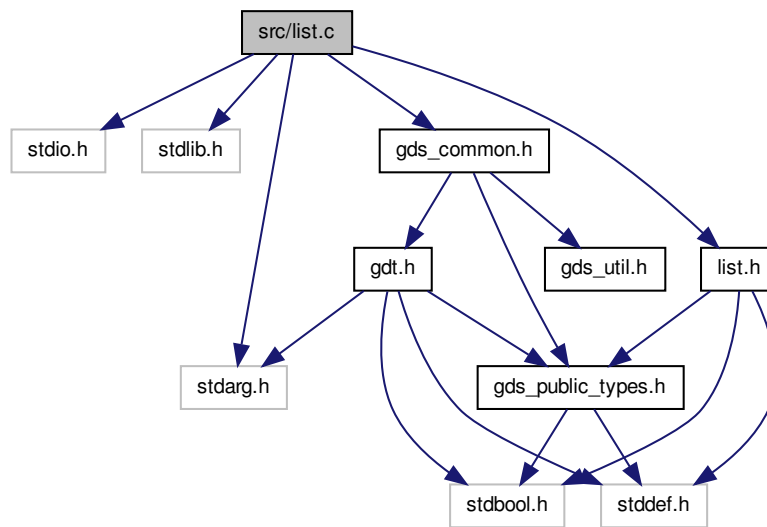
0	First value is equal to second value
-1	First value is less than second value
1	First value is greater than second value

8.18 `src/list.c` File Reference

Implementation of generic list data structure.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include "gds_common.h"
#include "list.h"
```

Include dependency graph for list.c:



Data Structures

- struct [list_node](#)
- struct [list](#)

Typedefs

- typedef struct [list_node](#) * [ListNode](#)

Functions

- static [ListNode](#) [list_node_create](#) ([List](#) list, va_list ap)
Private function to create list node.
- static void [list_node_destroy](#) ([List](#) list, [ListNode](#) node)
Destroys a list node.
- static [ListNode](#) [list_node_at_index](#) ([List](#) list, const size_t index)
Private function to return the node at a specified index.
- static bool [list_insert_internal](#) ([List](#) list, [ListNode](#) node, const size_t index)
Private function to insert a node into a list.
- [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_index` (`List list`, const `size_t index`)
Deletes the value at the specified index of the 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_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.

8.18.1 Detailed Description

Implementation of generic list data structure. The list is implemented as a double-ended, double-linked list.

Todo Implement sorting.
Implement iterators.

Author

Paul Griffiths

Copyright

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

8.18.2 Typedef Documentation

8.18.2.1 typedef struct list_node * ListNode

List node structure

8.18.3 Function Documentation

8.18.3.1 static bool list_insert_internal (List *list*, ListNode *node*, const size_t *index*) [static]

Private function to insert a node into a list.

Parameters

<i>list</i>	A pointer to the list.
<i>node</i>	A pointer to the node to insert.
<i>index</i>	The index at which to insert.

Return values

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

8.18.3.2 static ListNode list_node_at_index (List *list*, const size_t *index*) [static]

Private function to return the node at a specified index.

Parameters

<i>list</i>	A pointer to the list.
<i>index</i>	The index of the requested node.

Return values

<i>NULL</i>	Failure, index out of range
<i>non-NULL</i>	A pointer to the node at the specified index

8.18.3.3 static ListNode list_node_create (List *list*, va_list *ap*) [static]

Private function to create list node.

Parameters

<i>list</i>	A pointer to the list.
<i>ap</i>	A <i>va_list</i> containing the data value for the node. This should be of a type appropriate to the type set when creating the list.

Return values

<i>NULL</i>	Failure, dynamic memory allocation failed
<i>non-NULL</i>	A pointer to the new node

8.18.3.4 static void list_node_destroy (List *list*, ListNode *node*) [static]

Destroys a list node.

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.

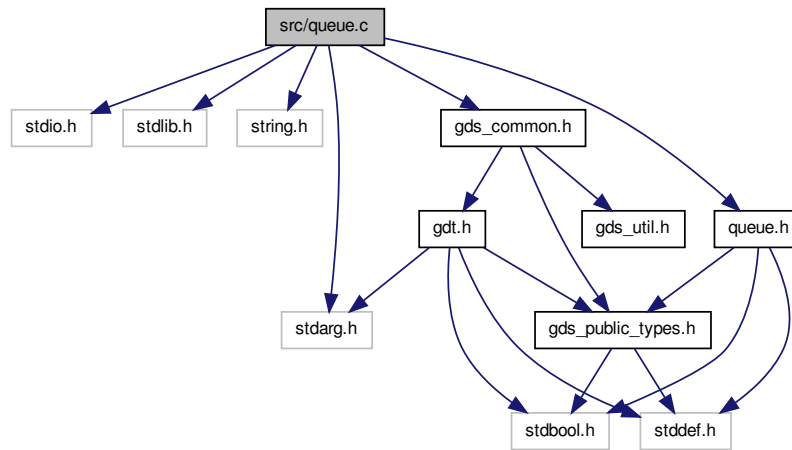
Parameters

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

8.19 src/queue.c File Reference

Implementation of generic queue data structure.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
#include "gds_common.h"
#include "queue.h"
Include dependency graph for queue.c:
```



Data Structures

- struct [queue](#)

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_free_space](#) (Queue queue)
Retrieves the free space on a queue.
- size_t [queue_size](#) (Queue queue)
Retrieves the current size of a queue.

Variables

- static const size_t `GROWTH` = 2

Growth factor for dynamic memory allocation.

8.19.1 Detailed Description

Implementation of 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/>

8.19.2 Variable Documentation

8.19.2.1 `const size_t GROWTH = 2` [static]

Growth factor for dynamic memory allocation.

Attention

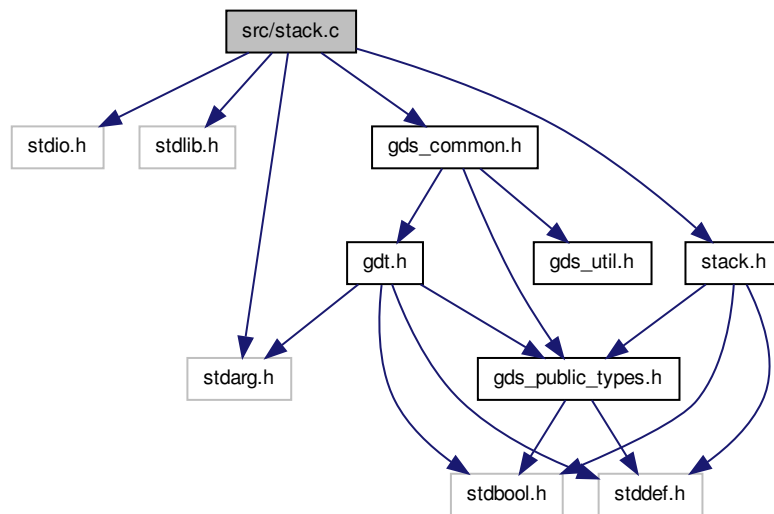
`queue_push()` relies on this being at least 2.

8.20 src/stack.c File Reference

Implementation of generic stack data structure.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include "gds_common.h"
#include "stack.h"
```

Include dependency graph for `stack.c`:



Data Structures

- struct [stack](#)

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_free_space](#) (Stack stack)
Retrieves the free space on a stack.
- size_t [stack_size](#) (Stack stack)
Retrieves the current size of a stack.

Variables

- static const size_t `GROWTH` = 2

8.20.1 Detailed Description

Implementation of 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/>

8.20.2 Variable Documentation

8.20.2.1 const size_t `GROWTH` = 2 [static]

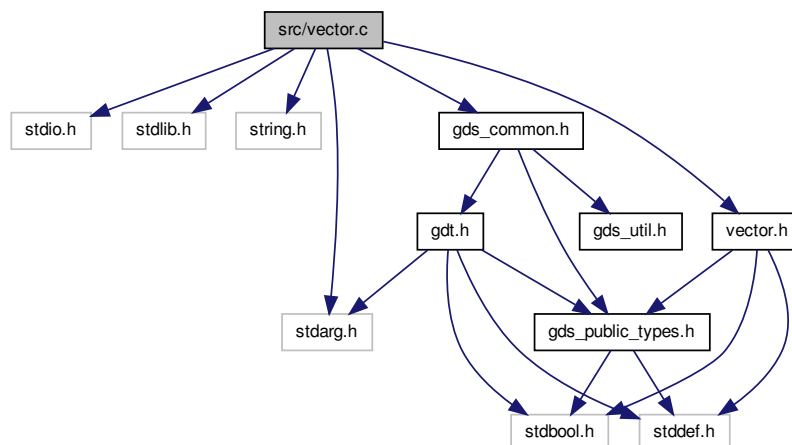
Growth factor for dynamic memory allocation

8.21 src/vector.c File Reference

Implementation of generic vector data structure.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
#include "gds_common.h"
#include "vector.h"
```

Include dependency graph for vector.c:



Data Structures

- struct [vector](#)

Functions

- static bool [vector_insert_internal](#) ([Vector vector](#), const size_t index, va_list ap)
Private function to insert a vector element.
- [Vector vector_create](#) (const size_t capacity, const enum [gds_datatype](#) type, const int opts,...)
Creates a new vector.
- void [vector_destroy](#) ([Vector vector](#))
Destroys a vector.
- bool [vector_append](#) ([Vector vector](#),...)
Appends a value to the back of a vector.
- bool [vector_prepend](#) ([Vector vector](#),...)
Prepends a value to the front of a vector.
- bool [vector_insert](#) ([Vector vector](#), const size_t index,...)
Inserts a value into a vector.
- bool [vector_delete_index](#) ([Vector vector](#), const size_t index)
Deletes the value at the specified index of the vector.
- bool [vector_delete_front](#) ([Vector vector](#))
Deletes the value at the front of the vector.
- bool [vector_delete_back](#) ([Vector vector](#))
Deletes the value at the back of the vector.
- bool [vector_element_at_index](#) ([Vector vector](#), const size_t index, void *p)
Gets the value at the specified index of the vector.
- bool [vector_set_element_at_index](#) ([Vector vector](#), const size_t index,...)
Sets the value at the specified index of the vector.
- bool [vector_find](#) ([Vector vector](#), size_t *index,...)
Tests if a value is contained in a vector.
- void [vector_sort](#) ([Vector vector](#))
Sorts a vector in-place, in ascending order.
- void [vector_reverse_sort](#) ([Vector vector](#))
Sorts a vector in-place, in descending order.
- bool [vector_is_empty](#) ([Vector vector](#))
Tests if a vector is empty.
- size_t [vector_length](#) ([Vector vector](#))
Returns the length of a vector.
- size_t [vector_capacity](#) ([Vector vector](#))
Returns the capacity of a vector.
- size_t [vector_free_space](#) ([Vector vector](#))
Returns the free space in a vector.

Variables

- static const size_t [GROWTH](#) = 2

8.21.1 Detailed Description

Implementation of 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/>

8.21.2 Function Documentation

8.21.2.1 `static bool vector_insert_internal (Vector vector, const size_t index, va_list ap)` `[static]`

Private function to insert a vector element.

Parameters

<i>vector</i>	A pointer to the vector.
<i>index</i>	The index at which to insert.
<i>ap</i>	A <code>va_list</code> containing the value to be inserted. This should be of a type appropriate to the type set when creating the vector.

Return values

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

8.21.3 Variable Documentation

8.21.3.1 `const size_t GROWTH = 2` `[static]`

Growth factor for dynamic memory allocation

Index

- back
 - queue, [42](#)
- c
 - gdt_generic_datatype, [37](#)
- capacity
 - queue, [42](#)
 - stack, [44](#)
 - vector, [45](#)
- compfunc
 - gdt_generic_datatype, [37](#)
 - list, [40](#)
 - vector, [45](#)
- d
 - gdt_generic_datatype, [38](#)
- DATATYPE_CHAR
 - Private functionality for manipulating generic datatypes, [12](#)
- DATATYPE_DOUBLE
 - Private functionality for manipulating generic datatypes, [12](#)
- DATATYPE_INT
 - Private functionality for manipulating generic datatypes, [12](#)
- DATATYPE_LONG
 - Private functionality for manipulating generic datatypes, [12](#)
- DATATYPE_LONG_LONG
 - Private functionality for manipulating generic datatypes, [12](#)
- DATATYPE_POINTER
 - Private functionality for manipulating generic datatypes, [12](#)
- DATATYPE_SIGNED_CHAR
 - Private functionality for manipulating generic datatypes, [12](#)
- DATATYPE_SIZE_T
 - Private functionality for manipulating generic datatypes, [12](#)
- DATATYPE_STRING
 - Private functionality for manipulating generic datatypes, [12](#)
- DATATYPE_UNSIGNED_CHAR
 - Private functionality for manipulating generic datatypes, [12](#)
- DATATYPE_UNSIGNED_INT
 - Private functionality for manipulating generic datatypes, [12](#)
- DATATYPE_UNSIGNED_LONG
 - Private functionality for manipulating generic datatypes, [12](#)
- DATATYPE_UNSIGNED_LONG_LONG
 - Private functionality for manipulating generic datatypes, [12](#)
- data
 - gdt_generic_datatype, [38](#)
- element
 - list_node, [41](#)
- elements
 - queue, [42](#)
 - stack, [44](#)
 - vector, [45](#)
- exit_on_error
 - list, [40](#)
 - queue, [42](#)
 - stack, [44](#)
 - vector, [45](#)
- free_on_destroy
 - list, [40](#)
 - queue, [42](#)
 - stack, [44](#)
 - vector, [45](#)
- front
 - queue, [43](#)
- GDS_EXIT_ON_ERROR
 - Public general generic data structures functionality, [15](#)
- GDS_FREE_ON_DESTROY
 - Public general generic data structures functionality, [15](#)
- GDS_RESIZABLE
 - Public general generic data structures functionality, [15](#)
- GROWTH
 - queue.c, [71](#)
 - stack.c, [73](#)
 - vector.c, [75](#)
- gds.dox, [47](#)
- gds_assert_quit
 - Public general generic data structures functionality, [15](#)
- gds_cfunc
 - Private functionality for manipulating generic datatypes, [12](#)
- gds_datatype
 - Private functionality for manipulating generic datatypes, [12](#)

- Private functionality for manipulating generic datatypes, [12](#)
- `gds_error_quit`
 - Public general generic data structures functionality, [15](#)
- `gds_option`
 - Public general generic data structures functionality, [15](#)
- `gds_strerror_quit`
 - Public general generic data structures functionality, [16](#)
- `gdt.c`
 - `gdt_compare_char`, [63](#)
 - `gdt_compare_double`, [63](#)
 - `gdt_compare_int`, [63](#)
 - `gdt_compare_long`, [64](#)
 - `gdt_compare_longlong`, [64](#)
 - `gdt_compare_schar`, [64](#)
 - `gdt_compare_sizet`, [65](#)
 - `gdt_compare_string`, [65](#)
 - `gdt_compare_uchar`, [65](#)
 - `gdt_compare_uint`, [65](#)
 - `gdt_compare_ulong`, [66](#)
 - `gdt_compare_ulonglong`, [66](#)
- `gdt_compare`
 - Private functionality for manipulating generic datatypes, [12](#)
- `gdt_compare_char`
 - `gdt.c`, [63](#)
- `gdt_compare_double`
 - `gdt.c`, [63](#)
- `gdt_compare_int`
 - `gdt.c`, [63](#)
- `gdt_compare_long`
 - `gdt.c`, [64](#)
- `gdt_compare_longlong`
 - `gdt.c`, [64](#)
- `gdt_compare_schar`
 - `gdt.c`, [64](#)
- `gdt_compare_sizet`
 - `gdt.c`, [65](#)
- `gdt_compare_string`
 - `gdt.c`, [65](#)
- `gdt_compare_uchar`
 - `gdt.c`, [65](#)
- `gdt_compare_uint`
 - `gdt.c`, [65](#)
- `gdt_compare_ulong`
 - `gdt.c`, [66](#)
- `gdt_compare_ulonglong`
 - `gdt.c`, [66](#)
- `gdt_compare_void`
 - Private functionality for manipulating generic datatypes, [12](#)
- `gdt_free`
 - Private functionality for manipulating generic datatypes, [13](#)
- `gdt_generic_datatype`, [37](#)
 - `c`, [37](#)
 - `compfunc`, [37](#)
 - `d`, [38](#)
 - `data`, [38](#)
 - `i`, [38](#)
 - `l`, [38](#)
 - `ll`, [38](#)
 - `p`, [38](#)
 - `pc`, [38](#)
 - `sc`, [38](#)
 - `st`, [38](#)
 - `type`, [38](#)
 - `uc`, [38](#)
 - `ui`, [38](#)
 - `ul`, [39](#)
 - `ull`, [39](#)
- `gdt_get_value`
 - Private functionality for manipulating generic datatypes, [13](#)
- `gdt_reverse_compare_void`
 - Private functionality for manipulating generic datatypes, [13](#)
- `gdt_set_value`
 - Private functionality for manipulating generic datatypes, [14](#)
- `head`
 - `list`, [40](#)
- `i`
 - `gdt_generic_datatype`, [38](#)
- `include/private/gds_common.h`, [47](#)
- `include/private/gdt.dox`, [48](#)
- `include/private/gdt.h`, [48](#)
- `include/public/gds_public_types.h`, [50](#)
- `include/public/gds_util.h`, [51](#)
- `include/public/general.dox`, [52](#)
- `include/public/list.dox`, [52](#)
- `include/public/list.h`, [52](#)
- `include/public/queue.dox`, [54](#)
- `include/public/queue.h`, [54](#)
- `include/public/stack.dox`, [56](#)
- `include/public/stack.h`, [56](#)
- `include/public/vector.dox`, [58](#)
- `include/public/vector.h`, [58](#)
- `l`
 - `gdt_generic_datatype`, [38](#)
- `length`
 - `list`, [40](#)
 - `vector`, [46](#)
- `List`
 - Public interface to generic list data structure, [17](#)
- `list`, [39](#)
 - `compfunc`, [40](#)
 - `exit_on_error`, [40](#)
 - `free_on_destroy`, [40](#)
 - `head`, [40](#)
 - `length`, [40](#)

- tail, 40
- type, 40
- list.c
 - list_insert_internal, 68
 - list_node_at_index, 69
 - list_node_create, 69
 - list_node_destroy, 69
 - ListNode, 68
- list_append
 - Public interface to generic list data structure, 18
- list_create
 - Public interface to generic list data structure, 18
- list_delete_back
 - Public interface to generic list data structure, 18
- list_delete_front
 - Public interface to generic list data structure, 18
- list_delete_index
 - Public interface to generic list data structure, 19
- list_destroy
 - Public interface to generic list data structure, 19
- list_element_at_index
 - Public interface to generic list data structure, 19
- list_find
 - Public interface to generic list data structure, 19
- list_insert
 - Public interface to generic list data structure, 20
- list_insert_internal
 - list.c, 68
- list_is_empty
 - Public interface to generic list data structure, 20
- list_length
 - Public interface to generic list data structure, 20
- list_node, 41
 - element, 41
 - next, 41
 - prev, 41
- list_node_at_index
 - list.c, 69
- list_node_create
 - list.c, 69
- list_node_destroy
 - list.c, 69
- list_prepend
 - Public interface to generic list data structure, 20
- list_set_element_at_index
 - Public interface to generic list data structure, 21
- ListNode
 - list.c, 68
- ll
 - gdt_generic_datatype, 38
- next
 - list_node, 41
- p
 - gdt_generic_datatype, 38
- pc
 - gdt_generic_datatype, 38
- prev
 - list_node, 41
- Private functionality for manipulating generic datatypes, 11
 - DATATYPE_CHAR, 12
 - DATATYPE_DOUBLE, 12
 - DATATYPE_INT, 12
 - DATATYPE_LONG, 12
 - DATATYPE_LONG_LONG, 12
 - DATATYPE_POINTER, 12
 - DATATYPE_SIGNED_CHAR, 12
 - DATATYPE_SIZE_T, 12
 - DATATYPE_STRING, 12
 - DATATYPE_UNSIGNED_CHAR, 12
 - DATATYPE_UNSIGNED_INT, 12
 - DATATYPE_UNSIGNED_LONG, 12
 - DATATYPE_UNSIGNED_LONG_LONG, 12
 - gds_cfunc, 12
 - gds_datatype, 12
 - gdt_compare, 12
 - gdt_compare_void, 12
 - gdt_free, 13
 - gdt_get_value, 13
 - gdt_reverse_compare_void, 13
 - gdt_set_value, 14
- Public general generic data structures functionality, 15
 - GDS_EXIT_ON_ERROR, 15
 - GDS_FREE_ON_DESTROY, 15
 - GDS_RESIZABLE, 15
 - gds_assert_quit, 15
 - gds_error_quit, 15
 - gds_option, 15
 - gds_strerror_quit, 16
- Public interface to generic list data structure, 17
 - List, 17
 - list_append, 18
 - list_create, 18
 - list_delete_back, 18
 - list_delete_front, 18
 - list_delete_index, 19
 - list_destroy, 19
 - list_element_at_index, 19
 - list_find, 19
 - list_insert, 20
 - list_is_empty, 20
 - list_length, 20
 - list_prepend, 20
 - list_set_element_at_index, 21
- Public interface to generic queue data structure, 22
 - Queue, 22
 - queue_capacity, 22
 - queue_create, 23
 - queue_destroy, 23
 - queue_free_space, 23
 - queue_is_empty, 23
 - queue_is_full, 24
 - queue_peek, 24
 - queue_pop, 24
 - queue_push, 24

- queue_size, 25
- Public interface to generic stack data structure, 26
 - Stack, 26
 - stack_capacity, 26
 - stack_create, 27
 - stack_destroy, 27
 - stack_free_space, 27
 - stack_is_empty, 27
 - stack_is_full, 28
 - stack_peek, 28
 - stack_pop, 28
 - stack_push, 28
 - stack_size, 29
- Public interface to generic vector data structure., 30
 - Vector, 31
 - vector_append, 31
 - vector_capacity, 31
 - vector_create, 31
 - vector_delete_back, 32
 - vector_delete_front, 32
 - vector_delete_index, 32
 - vector_destroy, 32
 - vector_element_at_index, 33
 - vector_find, 33
 - vector_free_space, 33
 - vector_insert, 33
 - vector_is_empty, 34
 - vector_length, 34
 - vector_prepend, 34
 - vector_reverse_sort, 35
 - vector_set_element_at_index, 35
 - vector_sort, 35
- Queue
 - Public interface to generic queue data structure, 22
- queue, 42
 - back, 42
 - capacity, 42
 - elements, 42
 - exit_on_error, 42
 - free_on_destroy, 42
 - front, 43
 - resizable, 43
 - size, 43
 - type, 43
- queue.c
 - GROWTH, 71
- queue_capacity
 - Public interface to generic queue data structure, 22
- queue_create
 - Public interface to generic queue data structure, 23
- queue_destroy
 - Public interface to generic queue data structure, 23
- queue_free_space
 - Public interface to generic queue data structure, 23
- queue_is_empty
 - Public interface to generic queue data structure, 23
- queue_is_full
 - Public interface to generic queue data structure, 24
- queue_peek
 - Public interface to generic queue data structure, 24
- queue_pop
 - Public interface to generic queue data structure, 24
- queue_push
 - Public interface to generic queue data structure, 24
- queue_size
 - Public interface to generic queue data structure, 25
- resizable
 - queue, 43
 - stack, 44
- sc
 - gdt_generic_datatype, 38
- size
 - queue, 43
- src/gds_util.c, 60
- src/gdt.c, 61
- src/list.c, 66
- src/queue.c, 69
- src/stack.c, 71
- src/vector.c, 73
- st
 - gdt_generic_datatype, 38
- Stack
 - Public interface to generic stack data structure, 26
- stack, 43
 - capacity, 44
 - elements, 44
 - exit_on_error, 44
 - free_on_destroy, 44
 - resizable, 44
 - top, 44
 - type, 44
- stack.c
 - GROWTH, 73
- stack_capacity
 - Public interface to generic stack data structure, 26
- stack_create
 - Public interface to generic stack data structure, 27
- stack_destroy
 - Public interface to generic stack data structure, 27
- stack_free_space
 - Public interface to generic stack data structure, 27
- stack_is_empty
 - Public interface to generic stack data structure, 27
- stack_is_full
 - Public interface to generic stack data structure, 28
- stack_peek
 - Public interface to generic stack data structure, 28
- stack_pop
 - Public interface to generic stack data structure, 28
- stack_push
 - Public interface to generic stack data structure, 28
- stack_size
 - Public interface to generic stack data structure, 29
- tail

- list, [40](#)
- top
 - stack, [44](#)
- type
 - gdt_generic_datatype, [38](#)
 - list, [40](#)
 - queue, [43](#)
 - stack, [44](#)
 - vector, [46](#)
- uc
 - gdt_generic_datatype, [38](#)
- ui
 - gdt_generic_datatype, [38](#)
- ul
 - gdt_generic_datatype, [39](#)
- ull
 - gdt_generic_datatype, [39](#)
- Vector
 - Public interface to generic vector data structure., [31](#)
- vector, [45](#)
 - capacity, [45](#)
 - compfunc, [45](#)
 - elements, [45](#)
 - exit_on_error, [45](#)
 - free_on_destroy, [45](#)
 - length, [46](#)
 - type, [46](#)
- vector.c
 - GROWTH, [75](#)
 - vector_insert_internal, [75](#)
- vector_append
 - Public interface to generic vector data structure., [31](#)
- vector_capacity
 - Public interface to generic vector data structure., [31](#)
- vector_create
 - Public interface to generic vector data structure., [31](#)
- vector_delete_back
 - Public interface to generic vector data structure., [32](#)
- vector_delete_front
 - Public interface to generic vector data structure., [32](#)
- vector_delete_index
 - Public interface to generic vector data structure., [32](#)
- vector_destroy
 - Public interface to generic vector data structure., [32](#)
- vector_element_at_index
 - Public interface to generic vector data structure., [33](#)
- vector_find
 - Public interface to generic vector data structure., [33](#)
- vector_free_space
 - Public interface to generic vector data structure., [33](#)
- vector_insert
 - Public interface to generic vector data structure., [33](#)
- vector_insert_internal
 - vector.c, [75](#)
- vector_is_empty
 - Public interface to generic vector data structure., [34](#)
- vector_length
 - Public interface to generic vector data structure., [34](#)
- vector_prepend
 - Public interface to generic vector data structure., [34](#)
- vector_reverse_sort
 - Public interface to generic vector data structure., [35](#)
- vector_set_element_at_index
 - Public interface to generic vector data structure., [35](#)
- vector_sort
 - Public interface to generic vector data structure., [35](#)