gds

Generated by Doxygen 1.8.1.2

Tue Nov 11 2014 22:19:42

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 gds_error_quit 16
6.2.3.3 gds_strdup 16
6.2.3.4 gds_strerror_quit 16
6.3 Public interface to generic list data structure 17
6.3.1 Detailed Description 18
6.3.2 Typedef Documentation 18
6.3.2.1 List 18
6.3.2.2 ListItr 18
6.3.3 Function Documentation 18
6.3.3.1 list_append 18
6.3.3.2 list_create 18
6.3.3.3 list_delete_back 19
6.3.3.4 list_delete_front 19
6.3.3.5 list_delete_index 19
6.3.3.6 list_destroy 19
6.3.3.7 list_element_at_index 19
6.3.3.8 list_find 20
6.3.3.9 list_find_itr 20
6.3.3.10 list_get_value_itr 20
6.3.3.11 list_insert 21
6.3.3.12 list_is_empty 21
6.3.3.13 list_itr_first 21
6.3.3.14 list_itr_last 21
6.3.3.15 list_itr_next 22
6.3.3.16 list_itr_previous 22
6.3.3.17 list_length 22
6.3.3.18 list_prepend 22
6.3.3.19 list_reverse_sort 23
6.3.3.20 list_set_element_at_index 23
6.3.3.21 list_sort 23
6.4 Public interface to generic queue data structure 24
6.4.1 Detailed Description 24
6.4.2 Typedef Documentation 24
6.4.2.1 Queue 24
6.4.3 Function Documentation 24
6.4.3.1 queue_capacity 24
6.4.3.2 queue_create 25
6.4.3.3 queue_destroy 25
6.4.3.4 queue_free_space 25
6.4.3.5 queue_is_empty 25

6.4.3.6 queue_is_full 26
6.4.3.7 queue_peek 26
6.4.3.8 queue_pop 26
6.4.3.9 queue_push 27
6.4.3.10 queue_size 27
6.5 Public interface to generic stack data structure 28
6.5.1 Detailed Description 28
6.5.2 Typedef Documentation 28
6.5.2.1 Stack 28
6.5.3 Function Documentation 28
6.5.3.1 stack_capacity 28
6.5.3.2 stack_create 29
6.5.3.3 stack_destroy 29
6.5.3.4 stack_free_space 29
6.5.3.5 stack_is_empty 29
6.5.3.6 stack_is_full 30
6.5.3.7 stack_peek 30
6.5.3.8 stack_pop 30
6.5.3.9 stack_push 31
6.5.3.10 stack_size 31
6.6 General purpose string manipulation functions 32
6.6.1 Detailed Description 32
6.6.2 Function Documentation 32
6.6.2.1 gds_strdup 32
6.6.2.2 gds_strndup 33
6.6.2.3 gds_trim 33
6.6.2.4 gds_trim_left 33
6.6.2.5 gds_trim_line_ending 34
6.6.2.6 gds_trim_right 34
6.6.2.7 list_string_create 34
6.6.2.8 list_string_destroy 34
6.6.2.9 pair_string_copy 34
6.6.2.10 pair_string_create 35
6.6.2.11 pair_string_destroy 35
6.6.2.12 split_string 35
6.7 Public interface to generic vector data structure. 36
6.7.1 Detailed Description 36
6.7.2 Typedef Documentation 37
6.7.2.1 Vector 37
6.7.3 Function Documentation 37

6.7.3.1 vector_append 37
6.7.3.2 vector_capacity 37
6.7.3.3 vector_create 37
6.7.3.4 vector_delete_back 38
6.7.3.5 vector_delete_front 38
6.7.3.6 vector_delete_index 38
6.7.3.7 vector_destroy 38
6.7.3.8 vector_element_at_index 39
6.7.3.9 vector_find 39
6.7.3.10 vector_free_space 39
6.7.3.11 vector_insert 40
6.7.3.12 vector_is_empty 40
6.7.3.13 vector_length 40
6.7.3.14 vector_prepend 40
6.7.3.15 vector_reverse_sort 41
6.7.3.16 vector_set_element_at_index 41
6.7.3.17 vector_sort 41

7 Data Structure Documentation 43

7.1 dict Struct Reference 43
7.1.1 Detailed Description 44
7.1.2 Field Documentation 44
7.1.2.1 buckets 44
7.1.2.2 exit_on_error 44
7.1.2.3 free_on_destroy 44
7.1.2.4 num_buckets 44
7.1.2.5 type 44
7.2 gdt_generic_datatype Struct Reference 44
7.2.1 Detailed Description 45
7.2.2 Field Documentation 45
7.2.2.1 c 45
7.2.2.2 compfunc 45
7.2.2.3 d 45
7.2.2.4 data 45
7.2.2.5 i 45
7.2.2.6 l 45
7.2.2.7 ll 45
7.2.2.8 p 45
7.2.2.9 pc 45
7.2.2.10 sc 45

7.2.2.11 st 46
7.2.2.12 type 46
7.2.2.13 uc 46
7.2.2.14 ui 46
7.2.2.15 ul 46
7.2.2.16 ull 46
7.3 kvpair Struct Reference 46
7.3.1 Detailed Description 47
7.3.2 Field Documentation 47
7.3.2.1 key 47
7.3.2.2 value 47
7.4 list Struct Reference 47
7.4.1 Detailed Description 48
7.4.2 Field Documentation 48
7.4.2.1 compfunc 48
7.4.2.2 exit_on_error 48
7.4.2.3 free_on_destroy 48
7.4.2.4 head 48
7.4.2.5 length 48
7.4.2.6 tail 48
7.4.2.7 type 48
7.5 list_node Struct Reference 49
7.5.1 Detailed Description 49
7.5.2 Field Documentation 49
7.5.2.1 element 49
7.5.2.2 next 49
7.5.2.3 prev 49
7.6 list_string Struct Reference 49
7.6.1 Detailed Description 50
7.6.2 Field Documentation 50
7.6.2.1 list 50
7.6.2.2 size 50
7.7 pair_string Struct Reference 50
7.7.1 Detailed Description 50
7.7.2 Field Documentation 50
7.7.2.1 first 50
7.7.2.2 second 50
7.8 queue Struct Reference 51
7.8.1 Detailed Description 51
7.8.2 Field Documentation 51

7.8.2.1 back 51
7.8.2.2 capacity 51
7.8.2.3 elements 51
7.8.2.4 exit_on_error 51
7.8.2.5 free_on_destroy 52
7.8.2.6 front 52
7.8.2.7 resizable 52
7.8.2.8 size 52
7.8.2.9 type 52
7.9 stack Struct Reference 52
7.9.1 Detailed Description 53
7.9.2 Field Documentation 53
7.9.2.1 capacity 53
7.9.2.2 elements 53
7.9.2.3 exit_on_error 53
7.9.2.4 free_on_destroy 53
7.9.2.5 resizable 53
7.9.2.6 top 53
7.9.2.7 type 53
7.10 vector Struct Reference 54
7.10.1 Detailed Description 54
7.10.2 Field Documentation 54
7.10.2.1 capacity 54
7.10.2.2 compfunc 54
7.10.2.3 elements 54
7.10.2.4 exit_on_error 54
7.10.2.5 free_on_destroy 55
7.10.2.6 length 55
7.10.2.7 type 55

8 File Documentation 57
8.1 gds.dox File Reference 57
8.2 include/private/gds_common.h File Reference 57
8.2.1 Detailed Description 58
8.3 include/private/gdt.dox File Reference 58
8.4 include/private/gdt.h File Reference 58
8.4.1 Detailed Description 60
8.5 include/public/dict.h File Reference 60
8.5.1 Detailed Description 61
8.5.2 Typedef Documentation 61

8.5.2.1 Dict 61
8.5.3 Function Documentation 62
8.5.3.1 dict_create 62
8.5.3.2 dict_destroy 62
8.5.3.3 dict_has_key 62
8.5.3.4 dict_insert 62
8.5.3.5 dict_value_for_key 63
8.6 include/public/gds_public_types.h File Reference 63
8.6.1 Detailed Description 64
8.7 include/public/gds_util.h File Reference 64
8.7.1 Detailed Description 65
8.8 include/public/general.dox File Reference 65
8.9 include/public/list.dox File Reference 65
8.10 include/public/list.h File Reference 65
8.10.1 Detailed Description 67
8.11 include/public/queue.dox File Reference 68
8.12 include/public/queue.h File Reference 68
8.12.1 Detailed Description 69
8.13 include/public/stack.dox File Reference 69
8.14 include/public/stack.h File Reference 69
8.14.1 Detailed Description 71
8.15 include/public/string_util.dox File Reference 71
8.16 include/public/string_util.h File Reference 71
8.16.1 Detailed Description 72
8.17 include/public/vector.dox File Reference 73
8.18 include/public/vector.h File Reference 73
8.18.1 Detailed Description 74
8.19 src/dict.c File Reference 75
8.19.1 Detailed Description 76
8.19.2 Typedef Documentation 76
8.19.2.1 KVPair 76
8.19.3 Function Documentation 76
8.19.3.1 dict_buckets_create 76
8.19.3.2 dict_buckets_destroy 77
8.19.3.3 dict_create 77
8.19.3.4 dict_destroy 77
8.19.3.5 dict_has_key 77
8.19.3.6 dict_has_key_internal 77
8.19.3.7 dict_insert 78
8.19.3.8 dict_value_for_key 78

8.19.3.9 djb2hash . . . 78
8.19.3.10 kvpair_compare . . . 79
8.19.3.11 kvpair_create . . . 79
8.19.3.12 kvpair_destroy . . . 79
8.19.4 Variable Documentation . . . 79
8.19.4.1 BUCKETS . . . 80
8.20 src/gds_util.c File Reference . . . 80
8.20.1 Detailed Description . . . 80
8.21 src/gdt.c File Reference . . . 80
8.21.1 Detailed Description . . . 82
8.21.2 Function Documentation . . . 82
8.21.2.1 gdt_compare_char . . . 82
8.21.2.2 gdt_compare_double . . . 82
8.21.2.3 gdt_compare_int . . . 83
8.21.2.4 gdt_compare_long . . . 83
8.21.2.5 gdt_compare_longlong . . . 83
8.21.2.6 gdt_compare_schar . . . 84
8.21.2.7 gdt_compare_sizet . . . 84
8.21.2.8 gdt_compare_string . . . 84
8.21.2.9 gdt_compare_uchar . . . 84
8.21.2.10 gdt_compare_uint . . . 85
8.21.2.11 gdt_compare_ulong . . . 85
8.21.2.12 gdt_compare_ulonglong . . . 85
8.22 src/list.c File Reference . . . 86
8.22.1 Detailed Description . . . 87
8.22.2 Typedef Documentation . . . 88
8.22.2.1 ListNode . . . 88
8.22.3 Function Documentation . . . 88
8.22.3.1 list_insert_internal . . . 88
8.22.3.2 list_node_at_index . . . 88
8.22.3.3 list_node_create . . . 88
8.22.3.4 list_node_destroy . . . 89
8.23 src/queue.c File Reference . . . 89
8.23.1 Detailed Description . . . 90
8.23.2 Variable Documentation . . . 90
8.23.2.1 GROWTH . . . 90
8.24 src/stack.c File Reference . . . 90
8.24.1 Detailed Description . . . 92
8.24.2 Variable Documentation . . . 92
8.24.2.1 GROWTH . . . 92

8.25 src/string_util.c File Reference 92
8.25.1 Detailed Description 93
8.25.2 Function Documentation 93
8.25.2.1 list_string_resize 93
8.26 src/vector.c File Reference 94
8.26.1 Detailed Description 95
8.26.2 Function Documentation 95
8.26.2.1 vector_insert_internal 95
8.26.3 Variable Documentation 96
8.26.3.1 GROWTH 96

Chapter 1

Generic Data Structures Library

GDS is a C language generic data structures library.

Chapter 2

Todo List

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 24
Public interface to generic stack data structure 28
General purpose string manipulation functions 32
Public interface to generic vector data structure. 36

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

dict 43
gdt_generic_datatype
    Generic datatype structure 44
kvpair 46
list 47
list_node 49
list_string
    Structure to hold a list of strings 49
pair_string
    Structure to hold a string pair 50
queue 51
stack 52
vector 54

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 57
include/private/gdt.h
Interface to generic data element functionality 58
include/public/dict.h
Interface to generic dictionary data structure 60
include/public/gds_public_types.h
Common public types for generic data structures library 63
include/public/gds_util.h
Interface to general utility functions 64
include/public/list.h
Interface to generic list data structure 65
include/public/queue.h
Interface to generic queue data structure 68
include/public/stack.h
Interface to generic stack data structure 69
include/public/string_util.h
Interface to string utility functions 71
include/public/vector.h
Interface to generic vector data structure 73
src/dict.c
Implementation of generic dictionary data structure 75
src/gds_util.c
Implementation of general utility functions 80
src/gdt.c
Implementation of generic data element functionality 80
src/list.c
Implementation of generic list data structure 86
src/queue.c
Implementation of generic queue data structure 89
src/stack.c
Implementation of generic stack data structure 90
src/string_util.c
Implementation of string utility functions 92
src/vector.c
Implementation of generic vector data structure 94

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

d1	A pointer to the first generic datatype.
d2	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.

6.1.4.2 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

p1	A pointer to the first generic datatype.
p2	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

data	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

data	A pointer to the generic datatype.
p	A pointer containing the address of an object of type appropriate to the type of the generic datatype set by the last call to `gdt_set_value()`. 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

p1	A pointer to the first generic datatype.
p2	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

data	A pointer to the generic datatype.
type	The type of data for the datatype to contain.
cfunc	A pointer to a comparison function. This is ignored for all types other than `DATATYPE_POI-NTER`. For `DATATYPE_POINTER`, this should contain the address of a function of type `int (*)(const void *, const void *)` 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, `NULL` can be provided.
ap	A `va_list` containing a single argument of the type appropriate to `type`, 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().
- char ∗ gds_strdup (const char ∗str)

  Dynamically duplicates a string.

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

msg	The format string for the message to print. Format specifiers are the same as the `printf()` 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

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

6.2.3.3 char∗ gds_strdup (const char ∗ *str*)

Dynamically duplicates a string.

Provided in case POSIX `strdup()` is not available.

Parameters

str	The string to duplicate.

Return values

NULL	Failure, dynamic allocation failed
non-NULL	A pointer to the new string

6.2.3.4 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

msg	The format string for the message to print. Format specifiers are the same as the `printf()` 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.
- typedef struct list_node ∗ ListItr
  Opaque list iterator 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.
- ListItr list_find_itr (List list,...)
  Tests if a value is contained in a list.
- bool list_sort (List list)
  Sorts a list in-place, in ascending order.
- bool list_reverse_sort (List list)
  Sorts a list in-place, in descending order.
- ListItr list_itr_first (List list)
  Returns an iterator to the first element of the list.
- ListItr list_itr_last (List list)
  Returns an iterator to the last element of the list.
- ListItr list_itr_next (ListItr itr)
  Increments a list iterator.
- ListItr list_itr_previous (ListItr itr)
  Decrements a list iterator.
- void list_get_value_itr (ListItr itr, void ∗p)
  Retrieves a value from an iterator.
- 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.

6.3.2.2 typedef struct **list_node**∗ **ListItr**

Opaque list iterator 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

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

Return values

true	Success
false	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

type	The datatype for the list.
opts	The following options can be OR'd together: `GDS_FREE_ON_DESTROY` to automatically `free()` pointer members when they are deleted or when the list is destroyed; `GDS_EXIT_ON_ERROR` to print a message to the standard error stream and `exit()`, rather than returning a failure status.
...	If `type` is `DATATYPE_POINTER`, 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

NULL	List creation failed.
non-NULL	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

list	A pointer to the list.

Return values

true	Success
false	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

list	A pointer to the list.

Return values

true	Success
false	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

list	A pointer to the list.
index	The index of the value to delete.

Return values

true	Success
false	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

list	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

list	A pointer to the list.
index	The index of the value to get.
p	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

true	Success
false	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

list	A pointer to the list.
index	A pointer to a `size_t` 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

true	The value was found in the list
false	The value was not found in the list

6.3.3.9 ListItr list_find_itr (List *list,* ...)

Tests if a value is contained in a list.

Parameters

list	A pointer to 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

NULL	The value was not found in the list
non-NULL	A list iterator pointing to the first occurrence of the vaue in the list.

6.3.3.10 void list_get_value_itr (ListItr *itr,* void ∗ *p*)

Retrieves a value from an iterator.

Parameters

itr	A pointer to the iterator.
p	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 given iterator.

6.3.3.11 bool list_insert (List *list*, const size_t *index*, ...)

Inserts a value into a list.

Parameters

list	A pointer to the list.
index	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

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

6.3.3.12 bool list_is_empty (List *list*)

Tests if a list is empty.

Parameters

list	A pointer to the list.

Return values

true	The list is empty
false	The list is not empty

6.3.3.13 ListItr list_itr_first (List *list*)

Returns an iterator to the first element of the list.

Parameters

list	A pointer to the list

Return values

NULL	Failure, list is empty
non-NULL	An iterator to the first element of the list

6.3.3.14 ListItr list_itr_last (List *list*)

Returns an iterator to the last element of the list.

Parameters

list	A pointer to the list

Return values

NULL	Failure, list is empty
non-NULL	An iterator to the last element of the list

6.3.3.15 ListItr list_itr_next (ListItr *itr*)

Increments a list iterator.

Parameters

itr	A pointer to the iterator.

Return values

NULL	End of list, no next iterator
non-NULL	An iterator to the next element of the list

6.3.3.16 ListItr list_itr_previous (ListItr *itr*)

Decrements a list iterator.

Parameters

itr	A pointer to the iterator.

Return values

NULL	Start of list, no previous iterator
non-NULL	An iterator to the previous element of the list

6.3.3.17 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

list	A pointer to the list.

Returns

The length of the list.

6.3.3.18 bool list_prepend (List *list, ...*)

Prepends a value to the front of a list.

Parameters

list	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

true	Success
false	Failure, dynamic memory allocation failed.

6.3.3.19 bool list_reverse_sort (List *list*)

Sorts a list in-place, in descending order.

Parameters

list	A pointer to the list.

Return values

true	Success
false	Failure, dynamic memory allocation failed.

6.3.3.20 bool list_set_element_at_index (List *list,* const size_t *index,* *...*)

Sets the value at the specified index of the list.

Parameters

list	A pointer to the list.
index	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

true	Success
false	Failure, index was out of range.

6.3.3.21 bool list_sort (List *list*)

Sorts a list in-place, in ascending order.

Parameters

list	A pointer to the list.

Return values

true	Success
false	Failure, dynamic memory allocation failed.

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

queue	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

capacity	The initial capacity of the queue.
type	The datatype for the queue.
opts	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

NULL	Queue creation failed.
non-NULL	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

queue	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

queue	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

queue	A pointer to the queue.

Return values

true	Queue is empty
false	Queue is not empty

6.4.3.6 bool queue_is_full (Queue *queue*)

Checks whether a queue is full.

Parameters

queue	A pointer to the queue.

Return values

true	Queue is full
false	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

queue	A pointer to the queue.
p	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

true	Success
false	Failure, queue is empty.

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

Pops a value from the queue.

Parameters

queue	A pointer to the queue.
p	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

true	Success
false	Failure, queue is empty.

6.4.3.9 bool queue_push (Queue *queue,* *...*)

Pushes a value onto the queue.

Parameters

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

Return values

true	Success
false	Failure, either because the queue is full or, if the `GDS_RESIZABLE` 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

queue	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

stack	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

capacity	The initial capacity of the stack.
type	The datatype for the stack.
opts	The following options can be OR'd together: `GDS_RESIZABLE` to dynamically resize the stack on-demand; `GDS_FREE_ON_DESTROY` to automatically `free()` pointer members when they are deleted or when the stack 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

NULL	Stack creation failed.
non-NULL	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

stack	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

stack	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

stack	A pointer to the stack.

Return values

true	Stack is empty
false	Stack is not empty

6.5.3.6 bool stack_is_full (Stack *stack*)

Checks whether a stack is full.

Parameters

stack	A pointer to the stack.

Return values

true	Stack is full
false	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

stack	A pointer to the stack.
p	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

true	Success
false	Failure, stack is empty.

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

Pops a value from the stack.

Parameters

stack	A pointer to the stack.
p	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

true	Success
false	Failure, stack is empty.

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

Pushes a value onto the stack.

Parameters

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

Return values

true	Success
false	Failure, either because the stack is full or, if the `GDS_RESIZABLE` 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

stack	A pointer to the stack.

Returns

The size of the stack.

6.6 General purpose string manipulation functions

Data Structures

- struct pair_string

  Structure to hold a string pair.
- struct list_string

  Structure to hold a list of strings.

Functions

- char ∗ gds_trim_line_ending (char ∗str)

  Trims CR and LF characters from the end of a string.
- char ∗ gds_trim_right (char ∗str)

  Trims trailing whitespace from a string.
- char ∗ gds_trim_left (char ∗str)

  Trims leading whitespace from a string.
- char ∗ gds_trim (char ∗str)

  Trims leading and trailing whitespace from a string.
- char ∗ gds_strdup (const char ∗str)

  Duplicates a string.
- char ∗ gds_strndup (const char ∗str, const size_t n)

  Duplicates at most n characters of a string.
- struct pair_string ∗ pair_string_create (const char ∗str, const char delim)

  Splits a string into a string pair.
- struct pair_string ∗ pair_string_copy (const struct pair_string ∗pair)

  Copies a string pair.
- void pair_string_destroy (struct pair_string ∗pair)

  Destroys a string pair.
- struct list_string ∗ list_string_create (const size_t n)

  Creates a string list.
- struct list_string ∗ split_string (const char ∗str, const char delim)

  Splits a string into a string list.
- void list_string_destroy (struct list_string ∗list)

  Destroys a string list.

6.6.1 Detailed Description

This module contains general purpose functions for working with and manipulating C-style strings.

6.6.2 Function Documentation

6.6.2.1 char∗ gds_strdup (const char ∗ *str*)

Duplicates a string.

Parameters

str	The string to duplicate.

Return values

NULL	Failure, dynamic memory allocation failed
non-NULL	A pointer to the duplicated string

Duplicates a string.

Provided in case POSIX `strdup()` is not available.

Parameters

str	The string to duplicate.

Return values

NULL	Failure, dynamic allocation failed
non-NULL	A pointer to the new string

6.6.2.2 char∗ gds_strndup (const char ∗ *str,* const size_t *n*)

Duplicates at most n characters of a string.

Parameters

str	The string to duplicate.
n	The maximum number of characters to duplicate.

Return values

NULL	Failure, dynamic memory allocation failed
non-NULL	A pointer to the duplicated string

6.6.2.3 char∗ gds_trim (char ∗ *str*)

Trims leading and trailing whitespace from a string.

Parameters

str	The string to trim.

Returns

A pointer to the passed string.

6.6.2.4 char∗ gds_trim_left (char ∗ *str*)

Trims leading whitespace from a string.

Parameters

str	The string to trim.

Returns

A pointer to the passed string.

6.6.2.5 char∗ gds_trim_line_ending (char ∗ *str*)

Trims CR and LF characters from the end of a string.

Parameters

str	The string to trim.

Returns

A pointer to the passed string.

6.6.2.6 char∗ gds_trim_right (char ∗ *str*)

Trims trailing whitespace from a string.

Parameters

str	The string to trim.

Returns

A pointer to the passed string.

6.6.2.7 struct list_string∗ list_string_create (const size_t *n*) [read]

Creates a string list.

Parameters

n	The capacity of the string list.

Return values

NULL	Failure, dynamic memory allocation failed
non-NULL	A pointer to the new string list

6.6.2.8 void list_string_destroy (struct list_string ∗ *list*)

Destroys a string list.

Parameters

list	The string list to destroy.

6.6.2.9 struct pair_string∗ pair_string_copy (const struct pair_string ∗ *pair*) [read]

Copies a string pair.

Parameters

pair	The string pair to copy.

Return values

NULL	Failure, dynamic memory allocation failed
non-NULL	A pointer to the new string pair

6.6.2.10 struct pair_string∗ pair_string_create (const char ∗ *str*, const char *delim*) [read]

Splits a string into a string pair.

Parameters

str	The string to split.
delim	The character on which to split.

Return values

NULL	Failure, dynamic memory allocation failed
non-NULL	A pointer to the new string pair

6.6.2.11 void pair_string_destroy (struct pair_string ∗ *pair*)

Destroys a string pair.

Parameters

pair	The pair to destroy.

6.6.2.12 struct list_string∗ split_string (const char ∗ *str*, const char *delim*) [read]

Splits a string into a string list.

Parameters

str	The string to split.
delim	The delimiter character.

Return values

NULL	Failure, dynamic memory allocation failed
non-NULL	A pointer to the new string pair

6.7 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.

6.7.1 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).

6.7.2 Typedef Documentation

6.7.2.1 typedef struct vector∗ Vector

Opaque vector type definition.

6.7.3 Function Documentation

6.7.3.1 bool vector_append (Vector *vector,* *...*)

Appends a value to the back of a vector.

Parameters

vector	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

true	Success
false	Failure, dynamic memory allocation failed.

6.7.3.2 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

vector	A pointer to the vector.

Returns

The capacity of the vector.

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

Creates a new vector.

Parameters

capacity	The initial capacity for the vector.
type	The datatype for the vector.
opts	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 `type` is `DATATYPE_POINTER`, 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

NULL	Vector creation failed.
non-NULL	A pointer to the new vector.

6.7.3.4 bool vector_delete_back (Vector *vector*)

Deletes the value at the back of the vector.

Parameters

vector	A pointer to the vector.

Return values

true	Success
false	Failure, dynamic memory allocation failed.

6.7.3.5 bool vector_delete_front (Vector *vector*)

Deletes the value at the front of the vector.

Parameters

vector	A pointer to the vector.

Return values

true	Success
false	Failure, dynamic memory allocation failed.

6.7.3.6 bool vector_delete_index (Vector *vector,* const size_t *index*)

Deletes the value at the specified index of the vector.

Parameters

vector	A pointer to the vector.
index	The index of the value to delete.

Return values

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

6.7.3.7 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

vector	A pointer to the vector.

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

Gets the value at the specified index of the vector.

Parameters

vector	A pointer to the vector.
index	The index of the value to get.
p	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

true	Success
false	Failure, index was out of range.

6.7.3.9 bool vector_find (Vector *vector,* size_t ∗ *index,* ...)

Tests if a value is contained in a vector.

Parameters

vector	A pointer to the vector.
index	A pointer to a `size_t` 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.
...	The value for which to search. This should be of a type appropriate to the type set when creating the vector.

Return values

true	The value was found in the vector
false	The value was not found in the vector

6.7.3.10 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

vector	A pointer to the vector.

Returns

The free space in the vector.

6.7.3.11 bool vector_insert (**Vector** *vector,* const size_t *index,* *...*)

Inserts a value into a vector.

Parameters

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

Return values

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

6.7.3.12 bool vector_is_empty (**Vector** *vector*)

Tests if a vector is empty.

Parameters

vector	A pointer to the vector.

Return values

true	The vector is empty
false	The vector is not empty

6.7.3.13 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

vector	A pointer to the vector.

Returns

The length of the vector.

6.7.3.14 bool vector_prepend (**Vector** *vector,* *...*)

Prepends a value to the front of a vector.

Parameters

vector	A pointer to the vector.
...	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

true	Success
false	Failure, dynamic memory allocation failed.

6.7.3.15 void vector_reverse_sort (Vector *vector*)

Sorts a vector in-place, in descending order.

Parameters

vector	A pointer to the vector.

6.7.3.16 bool vector_set_element_at_index (Vector *vector,* const size_t *index,* *...*)

Sets the value at the specified index of the vector.

Parameters

vector	A pointer to the vector.
index	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

true	Success
false	Failure, index was out of range.

6.7.3.17 void vector_sort (Vector *vector*)

Sorts a vector in-place, in ascending order.

Parameters

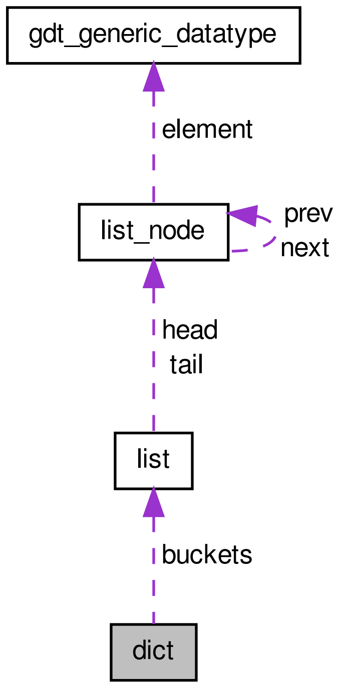
vector	A pointer to the vector.

Chapter 7

Data Structure Documentation

7.1 dict Struct Reference

Collaboration diagram for dict:



Data Fields

- size_t num_buckets
- List ∗ buckets
- enum gds_datatype type
- bool free_on_destroy
- bool exit_on_error

7.1.1 Detailed Description

Dict structure

7.1.2 Field Documentation

7.1.2.1 List∗ dict::buckets

The buckets

7.1.2.2 bool dict::exit_on_error

Exit on error if true

7.1.2.3 bool dict::free_on_destroy

Free pointer elements on destroy if true

7.1.2.4 size_t dict::num_buckets

Number of buckets

7.1.2.5 enum gds_datatype dict::type

Dict datatype

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

- src/dict.c

7.2 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
```

7.2.1 Detailed Description

Generic datatype structure.

7.2.2 Field Documentation

7.2.2.1 char gdt_generic_datatype::c

char

7.2.2.2 gds_cfunc gdt_generic_datatype::compfunc

Comparison function pointer

7.2.2.3 double gdt_generic_datatype::d

double

7.2.2.4 union { ... } gdt_generic_datatype::data

Data union

7.2.2.5 int gdt_generic_datatype::i

int

7.2.2.6 long gdt_generic_datatype::l

long

7.2.2.7 long long int gdt_generic_datatype::ll

long long

7.2.2.8 void* gdt_generic_datatype::p

void *

7.2.2.9 char* gdt_generic_datatype::pc

char *, string

7.2.2.10 signed char gdt_generic_datatype::sc

signed char

7.2.2.11 size_t gdt_generic_datatype::st

size_t

7.2.2.12 enum gds_datatype gdt_generic_datatype::type

Data type

7.2.2.13 unsigned char gdt_generic_datatype::uc

unsigned char

7.2.2.14 unsigned int gdt_generic_datatype::ui

unsigned int

7.2.2.15 unsigned long gdt_generic_datatype::ul

unsigned long

7.2.2.16 unsigned long long int gdt_generic_datatype::ull

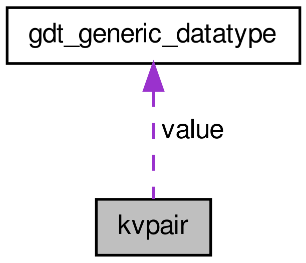
unsigned long long

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

- include/private/gdt.h

7.3 kvpair Struct Reference

Collaboration diagram for kvpair:



Data Fields

- char ∗ key
- struct gdt_generic_datatype value

7.3.1 Detailed Description

Key-Value pair structure

7.3.2 Field Documentation

7.3.2.1 char∗ kvpair::key

String key

7.3.2.2 struct gdt_generic_datatype kvpair::value

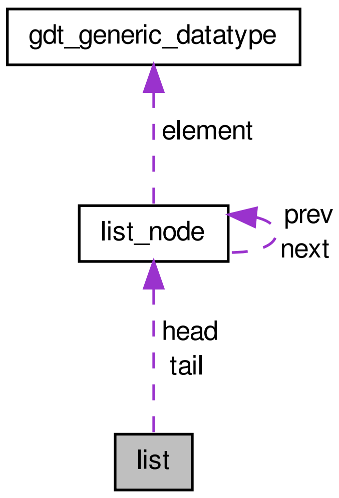
Generic datatype value

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

- src/dict.c

7.4 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

7.4.1 Detailed Description

List structure

7.4.2 Field Documentation

7.4.2.1 gds_cfunc list::compfunc

Element comparison function

7.4.2.2 bool list::exit_on_error

Exit on error if true

7.4.2.3 bool list::free_on_destroy

Free pointer elements on destroy if true

7.4.2.4 struct list_node* list::head

Pointer to head of list

7.4.2.5 size_t list::length

Length of list

7.4.2.6 struct list_node* list::tail

Pointer to tail of list

7.4.2.7 enum gds_datatype list::type

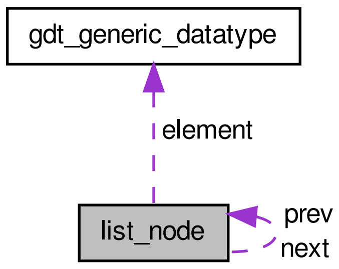
List datatype

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

- src/list.c

7.5 list_node Struct Reference

Collaboration diagram for list_node:



Data Fields

- struct gdt_generic_datatype element
- struct list_node ∗ prev
- struct list_node ∗ next

7.5.1 Detailed Description

List node structure

7.5.2 Field Documentation

7.5.2.1 struct **gdt_generic_datatype** list_node::element

Data element

7.5.2.2 struct **list_node**∗ list_node::next

Pointer to next node

7.5.2.3 struct **list_node**∗ list_node::prev

Pointer to previous node

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

- src/list.c

7.6 list_string Struct Reference

Structure to hold a list of strings.

```
#include <string_util.h>
```

Data Fields

- size_t size
- char ∗∗ list

7.6.1 Detailed Description

Structure to hold a list of strings.

7.6.2 Field Documentation

7.6.2.1 char∗∗ list_string::list

Pointer to the list

7.6.2.2 size_t list_string::size

Number of strings in the list

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

- include/public/string_util.h

7.7 pair_string Struct Reference

Structure to hold a string pair.

`#include <string_util.h>`

Data Fields

- char ∗ first
- char ∗ second

7.7.1 Detailed Description

Structure to hold a string pair.

7.7.2 Field Documentation

7.7.2.1 char∗ pair_string::first

First string of pair

7.7.2.2 char∗ pair_string::second

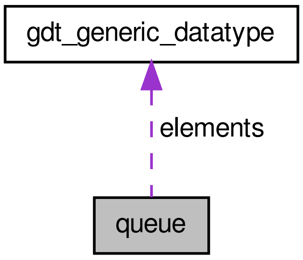
Second string of pair

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

- include/public/string_util.h

7.8 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

7.8.1 Detailed Description

Queue structure

7.8.2 Field Documentation

7.8.2.1 size_t queue::back

Back of queue

7.8.2.2 size_t queue::capacity

Capacity of queue

7.8.2.3 struct **gdt_generic_datatype**∗ queue::elements

Pointer to elements

7.8.2.4 bool queue::exit_on_error

Exit on error if true

7.8.2.5 bool queue::free_on_destroy

Free pointer elements on destroy if true

7.8.2.6 size_t queue::front

Front of queue

7.8.2.7 bool queue::resizable

Dynamically resizable if true

7.8.2.8 size_t queue::size

Size of queue

7.8.2.9 enum gds_datatype queue::type

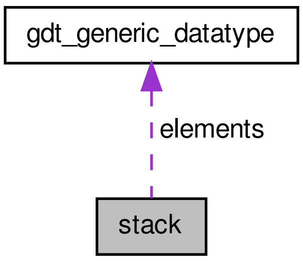
Queue datatype

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

- src/queue.c

7.9 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

7.9.1 Detailed Description

Stack structure

7.9.2 Field Documentation

7.9.2.1 size_t stack::capacity

Stack capacity

7.9.2.2 struct gdt_generic_datatype* stack::elements

Pointer to elements

7.9.2.3 bool stack::exit_on_error

Exit on error if true

7.9.2.4 bool stack::free_on_destroy

Free pointer elements on destroy if true

7.9.2.5 bool stack::resizable

Dynamically resizabe if true

7.9.2.6 size_t stack::top

Top of stack

7.9.2.7 enum gds_datatype stack::type

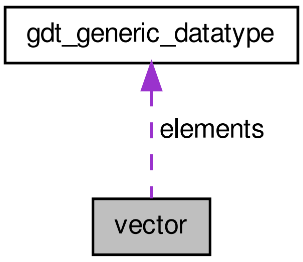
Stack datatype

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

- src/stack.c

7.10 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

7.10.1 Detailed Description

Vector structure

7.10.2 Field Documentation

7.10.2.1 size_t vector::capacity

Vector capacity

7.10.2.2 int(∗ vector::compfunc)(const void ∗, const void ∗)

Compare function

7.10.2.3 struct **gdt_generic_datatype**∗ vector::elements

Pointer to elements

7.10.2.4 bool vector::exit_on_error

Exit on error if true

7.10.2.5 bool vector::free_on_destroy

Free pointer elements on destroy if true

7.10.2.6 size_t vector::length

Vector length

7.10.2.7 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

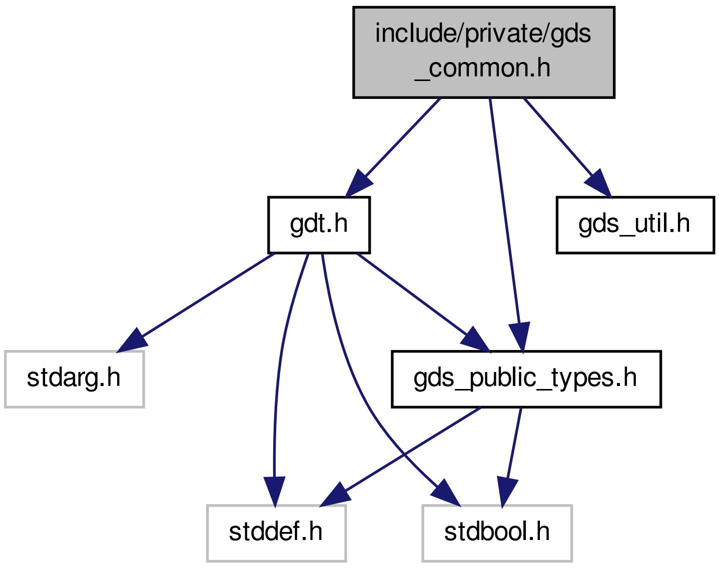
8.1 gds.dox File Reference

8.2 include/private/gds_common.h File Reference

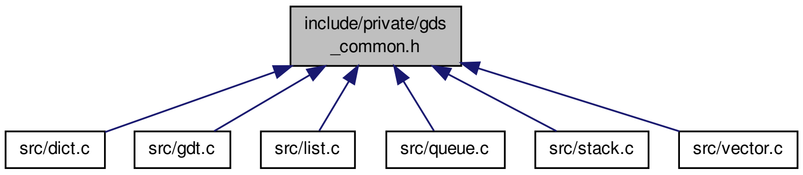
Common internal headers for data structures.

```
#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/

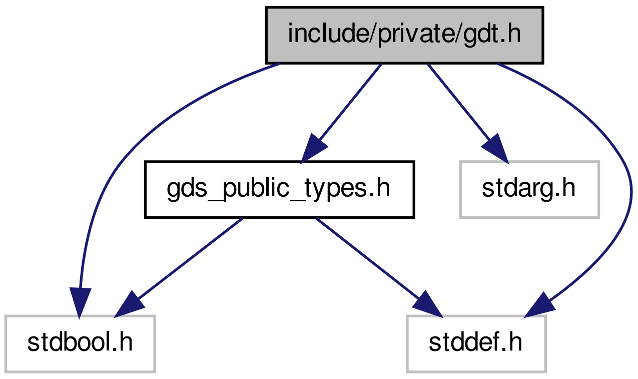
8.3 include/private/gdt.dox File Reference

8.4 include/private/gdt.h File Reference

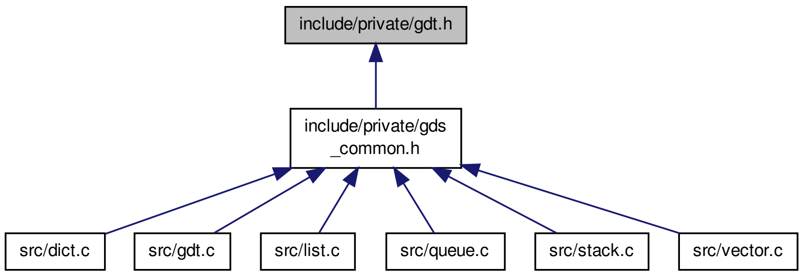
Interface to generic data element functionality.

```
#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

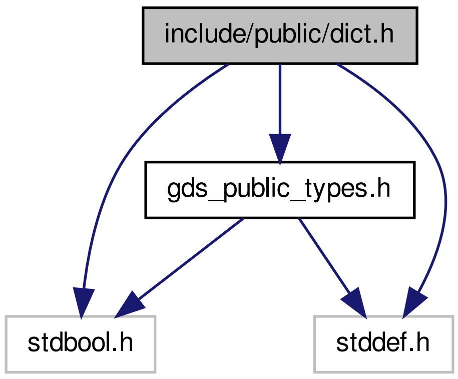
Copyright 2014 Paul Griffiths. Distributed under the terms of the GNU General Public License. http://www.gnu.org/licenses/

8.5 include/public/dict.h File Reference

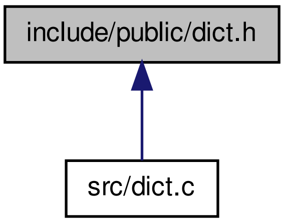
Interface to generic dictionary data structure.

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

Include dependency graph for dict.h:



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



Typedefs

- typedef struct dict ∗ Dict

  Opaque dictionary type definition.

Functions

- Dict dict_create (const enum gds_datatype type, const int opts)

  Creates a new dictionary.
- void dict_destroy (Dict dict)

  Destroys a dictionary.
- bool dict_insert (Dict dict, const char ∗key,...)

  Inserts a key-value into a dictionary.
- bool dict_has_key (Dict dict, const char ∗key)

  Checks whether a key exists in a dictionary.
- bool dict_value_for_key (Dict dict, const char ∗key, void ∗p)

  Retrieves the value for a key in the dictionary.

8.5.1 Detailed Description

Interface to generic dictionary data structure.

Author

Paul Griffiths

Copyright

8.5.2 Typedef Documentation

8.5.2.1 typedef struct dict∗ Dict

Opaque dictionary type definition.

8.5.3 Function Documentation

8.5.3.1 Dict dict_create (const enum gds_datatype *type*, const int *opts*)

Creates a new dictionary.

Parameters

type	The datatype for the dictionary.
opts	The following options can be OR'd together: `GDS_FREE_ON_DESTROY` to automatically `free()` pointer members when they are deleted or when the dictionary 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

NULL	Dictionart creation failed.
non-NULL	A pointer to the new dictionary.

8.5.3.2 void dict_destroy (Dict *dict*)

Destroys a dictionary.

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

Parameters

dict	A pointer to the dictionary.

8.5.3.3 bool dict_has_key (Dict *dict*, const char ∗ *key*)

Checks whether a key exists in a dictionary.

Parameters

dict	A pointer to the dictionary.
key	The key for which to search.

Return values

true	The key exists in the dictionary
false	The key does not exist in the dictionary

8.5.3.4 bool dict_insert (Dict *dict*, const char ∗ *key*, ...)

Inserts a key-value into a dictionary.

If the key already exists in the dictionary, the existing value will be overwritten. If `GDS_FREE_ON_DESTROY` was specified during dictionary creation, the existing element will be `free()`d prior to overwriting it.

Parameters

dict	A pointer to the dictionary.
key	The key.

...	The value corresponding to the key. This should be of a type appropriate to the type set when creating the dictionary.

Return values

true	Success
false	Failure, dynamic memory allocation failed

8.5.3.5 bool dict_value_for_key (Dict *dict,* const char ∗ *key,* void ∗ *p*)

Retrieves the value for a key in the dictionary.

Parameters

dict	A pointer to the dictionary.
key	The key for which to retrieve the value.
p	A pointer to an object of a type appropriate to the type set when creating the dictionary. The object at this address will be modified to contain the value for the specified key.

Return values

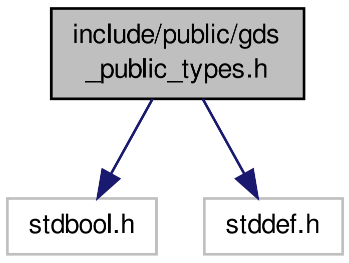
true	Success
false	Failure, key was not found

8.6 include/public/gds_public_types.h File Reference

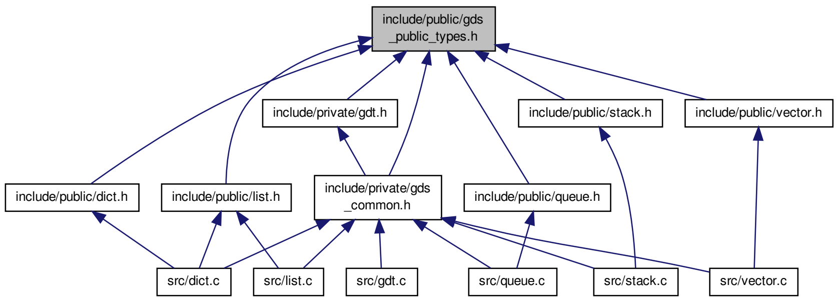
Common public types for generic data structures library.

```
#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_STRIN-G,
  DATATYPE_POINTER }

    Enumeration type for data element type.

8.6.1 Detailed Description

Common public types for generic data structures library.

Author

Paul Griffiths

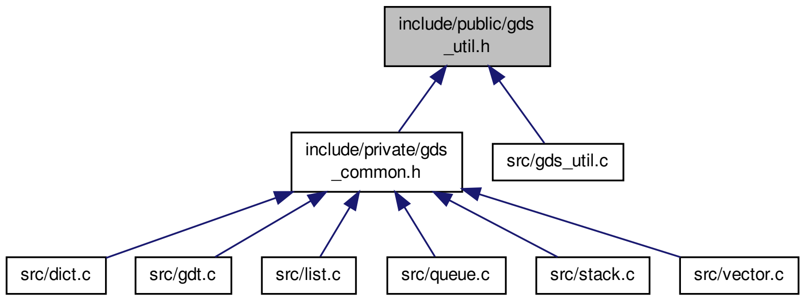
Copyright

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

8.7 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().
- char ∗ gds_strdup (const char ∗str)

  Dynamically duplicates a string.

8.7.1 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/`

8.8 include/public/general.dox File Reference

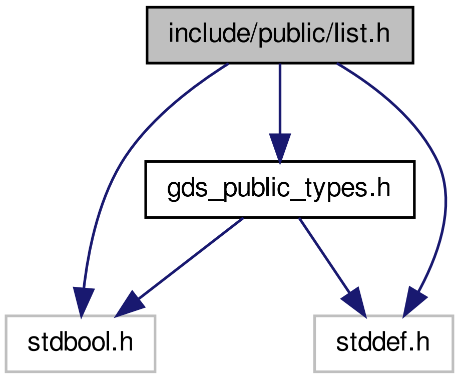
8.9 include/public/list.dox File Reference

8.10 include/public/list.h File Reference

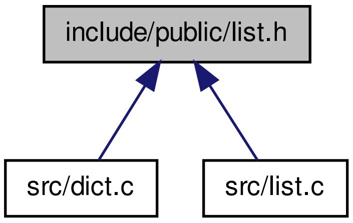
Interface to generic list data structure.

```
#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.
- typedef struct list_node ∗ ListItr

    Opaque list iterator 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.
- ListItr list_find_itr (List list,...)

  Tests if a value is contained in a list.
- bool list_sort (List list)

  Sorts a list in-place, in ascending order.
- bool list_reverse_sort (List list)

  Sorts a list in-place, in descending order.
- ListItr list_itr_first (List list)

  Returns an iterator to the first element of the list.
- ListItr list_itr_last (List list)

  Returns an iterator to the last element of the list.
- ListItr list_itr_next (ListItr itr)

  Increments a list iterator.
- ListItr list_itr_previous (ListItr itr)

  Decrements a list iterator.
- void list_get_value_itr (ListItr itr, void ∗p)

  Retrieves a value from an iterator.
- bool list_is_empty (List list)

  Tests if a list is empty.
- size_t list_length (List list)

  Returns the length of a list.

8.10.1 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/

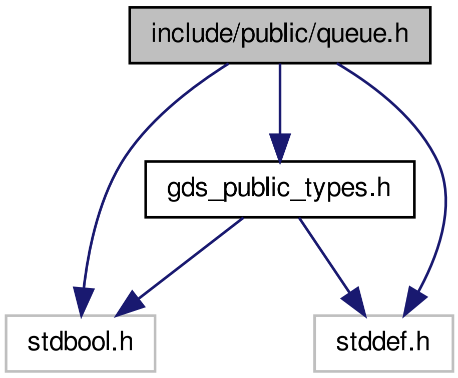
8.11 include/public/queue.dox File Reference

8.12 include/public/queue.h File Reference

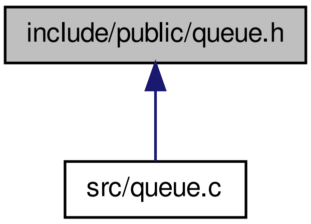
Interface to generic queue data structure.

```
#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.

8.12.1 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/

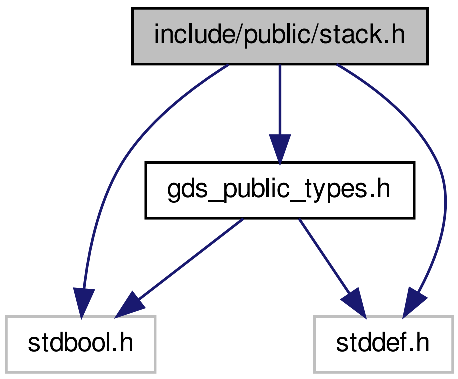
8.13 include/public/stack.dox File Reference

8.14 include/public/stack.h File Reference

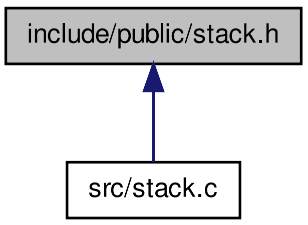
Interface to generic stack data structure.

```
#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.

8.14.1 Detailed Description

Interface to generic stack data structure.

Author

Paul Griffiths

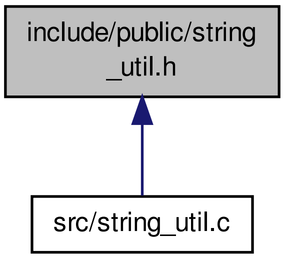
Copyright

8.15 include/public/string_util.dox File Reference

8.16 include/public/string_util.h File Reference

Interface to string utility functions.

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



Data Structures

- struct pair_string

  Structure to hold a string pair.
- struct list_string

  Structure to hold a list of strings.

Functions

- char ∗ gds_trim_line_ending (char ∗str)

  Trims CR and LF characters from the end of a string.
- char ∗ gds_trim_right (char ∗str)

  Trims trailing whitespace from a string.
- char ∗ gds_trim_left (char ∗str)

  Trims leading whitespace from a string.
- char ∗ gds_trim (char ∗str)

  Trims leading and trailing whitespace from a string.
- char ∗ gds_strdup (const char ∗str)

  Duplicates a string.
- char ∗ gds_strndup (const char ∗str, const size_t n)

  Duplicates at most n characters of a string.
- struct pair_string ∗ pair_string_create (const char ∗str, const char delim)

  Splits a string into a string pair.
- struct pair_string ∗ pair_string_copy (const struct pair_string ∗pair)

  Copies a string pair.
- void pair_string_destroy (struct pair_string ∗pair)

  Destroys a string pair.
- struct list_string ∗ list_string_create (const size_t n)

  Creates a string list.
- struct list_string ∗ split_string (const char ∗str, const char delim)

  Splits a string into a string list.
- void list_string_destroy (struct list_string ∗list)

  Destroys a string list.

8.16.1 Detailed Description

Interface to string utility functions.

Author

Paul Griffiths

Copyright

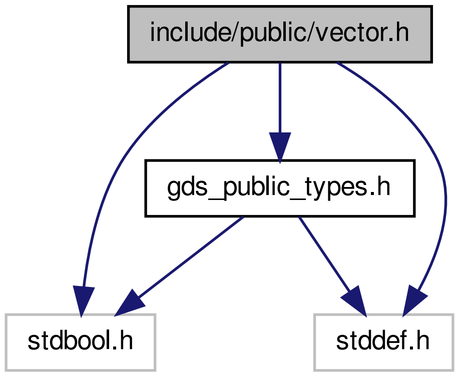
8.17 include/public/vector.dox File Reference

8.18 include/public/vector.h File Reference

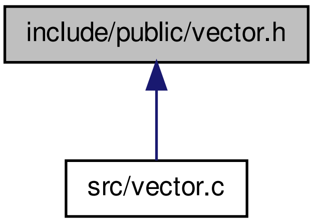
Interface to generic vector data structure.

```
#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.

8.18.1 Detailed Description

Interface to generic vector data structure.

Author

Paul Griffiths

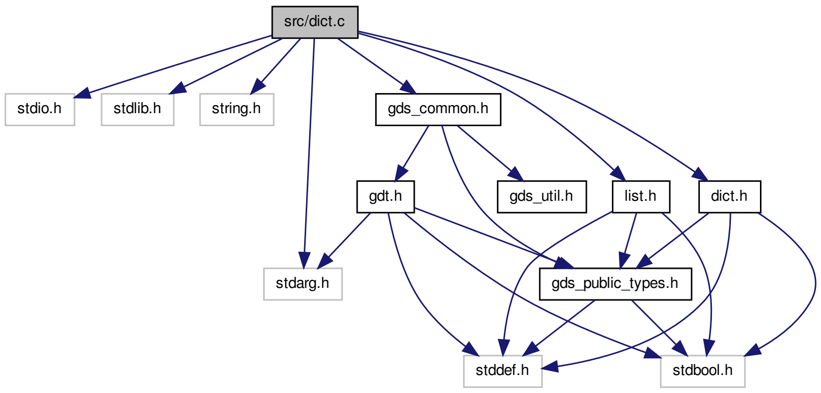
Copyright

8.19 src/dict.c File Reference

Implementation of generic dictionary data structure.

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

Include dependency graph for dict.c:



Data Structures

- struct kvpair
- struct dict

Typedefs

- typedef struct kvpair ∗ KVPair

Functions

- static KVPair kvpair_create (const char ∗key, const enum gds_datatype type, va_list ap)

    Creates a new key-value pair.
- static void kvpair_destroy (KVPair pair, const bool free_value)

    Destroys a key-value pair.
- static int kvpair_compare (const void ∗p1, const void ∗p2)

    Compares two key-value pairs by key.
- static bool dict_has_key_internal (Dict dict, const char ∗key, KVPair ∗pair)

    Internal function to check for the existence of a key.
- static bool dict_buckets_create (Dict dict)

    Helper function to create the dictionary buckets.
- static void dict_buckets_destroy (Dict dict)

    Helper function to destroy the dictionary buckets.

- static size_t djb2hash (const char ∗str)

  Calculates a hash of a string.
- Dict dict_create (const enum gds_datatype type, const int opts)

  Creates a new dictionary.
- void dict_destroy (Dict dict)

  Destroys a dictionary.
- bool dict_has_key (Dict dict, const char ∗key)

  Checks whether a key exists in a dictionary.
- bool dict_insert (Dict dict, const char ∗key,...)

  Inserts a key-value into a dictionary.
- bool dict_value_for_key (Dict dict, const char ∗key, void ∗p)

  Retrieves the value for a key in the dictionary.

Variables

- static const size_t BUCKETS = 256

8.19.1 Detailed Description

Implementation of generic dictionary 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 Typedef Documentation

8.19.2.1 typedef struct kvpair ∗ KVPair

Key-Value pair structure

8.19.3 Function Documentation

8.19.3.1 static bool dict_buckets_create (Dict *dict*) `[static]`

Helper function to create the dictionary buckets.

Parameters

dict	A pointer to the dictionary.

Return values

true	Success
false	Failure, dynamic memory allocation failed.

8.19.3.2 static void dict_buckets_destroy (**Dict** *dict*) `[static]`

Helper function to destroy the dictionary buckets.

Parameters

dict	A pointer to the dictionary.

8.19.3.3 Dict dict_create (const enum **gds_datatype** *type,* const int *opts*)

Creates a new dictionary.

Parameters

type	The datatype for the dictionary.
opts	The following options can be OR'd together: `GDS_FREE_ON_DESTROY` to automatically `free()` pointer members when they are deleted or when the dictionary 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

NULL	Dictionart creation failed.
non-NULL	A pointer to the new dictionary.

8.19.3.4 void dict_destroy (**Dict** *dict*)

Destroys a dictionary.

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

Parameters

dict	A pointer to the dictionary.

8.19.3.5 bool dict_has_key (**Dict** *dict,* const char ∗ *key*)

Checks whether a key exists in a dictionary.

Parameters

dict	A pointer to the dictionary.
key	The key for which to search.

Return values

true	The key exists in the dictionary
false	The key does not exist in the dictionary

8.19.3.6 static bool dict_has_key_internal (**Dict** *dict,* const char ∗ *key,* **KVPair** ∗ *pair*) `[static]`

Internal function to check for the existence of a key.

If the key is present, `pair` will be modified to contain the address of the key-value pair containing it.

Parameters

dict	A pointer to the dictionary.
key	The key for which to search.
pair	A pointer to a key-value pair pointer. If the key is found, the pointer at this address will be modified to contain the address of the pair containing the key.

Return values

true	Key was found
false	Key was not found

8.19.3.7 bool dict_insert (Dict *dict*, const char ∗ *key*, ...)

Inserts a key-value into a dictionary.

If the key already exists in the dictionary, the existing value will be overwritten. If `GDS_FREE_ON_DESTROY` was specified during dictionary creation, the existing element will be `free()`d prior to overwriting it.

Parameters

dict	A pointer to the dictionary.
key	The key.
...	The value corresponding to the key. This should be of a type appropriate to the type set when creating the dictionary.

Return values

true	Success
false	Failure, dynamic memory allocation failed

8.19.3.8 bool dict_value_for_key (Dict *dict*, const char ∗ *key*, void ∗ *p*)

Retrieves the value for a key in the dictionary.

Parameters

dict	A pointer to the dictionary.
key	The key for which to retrieve the value.
p	A pointer to an object of a type appropriate to the type set when creating the dictionary. The object at this address will be modified to contain the value for the specified key.

Return values

true	Success
false	Failure, key was not found

8.19.3.9 static size_t djb2hash (const char ∗ *str*) `[static]`

Calculates a hash of a string.

Uses Dan Bernstein's djb2 algorithm.

Parameters

str	A pointer to a string

Returns

The hash value

8.19.3.10 static int kvpair_compare (const void ∗ *p1*, const void ∗ *p2*) `[static]`

Compares two key-value pairs by key.

This function is suitable for passing to qsort().

Parameters

p1	A pointer to the first pair.
p2	A pointer to the second pair.

Return values

0	The keys of the two pairs are equal
-1	The key of the first pair is less than the key of the second pair
1	The key of the first pair is greater than the key of the second pair

8.19.3.11 static KVPair kvpair_create (const char ∗ *key*, const enum gds_datatype *type*, va_list *ap*) `[static]`

Creates a new key-value pair.

Parameters

key	The key for the new pair.
type	The datatype for the new pair
ap	A `va_list` containing the data value for the pair. This should be of a type appropriate to the type set when creating the list.

Return values

NULL	Failure, dynamic memory allocation failed
non-NULL	Success

8.19.3.12 static void kvpair_destroy (KVPair *pair*, const bool *free_value*) `[static]`

Destroys a key-value pair.

Parameters

pair	A pointer to the pair to destroy.
free_value	If true, the data will be passed to gdt_free()

8.19.4 Variable Documentation

8.19.4.1 const size_t BUCKETS = 256 [static]

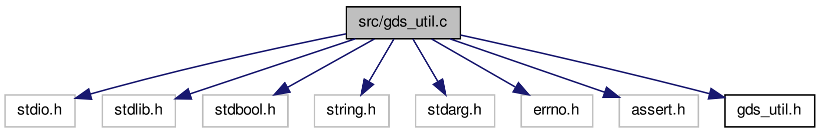
Number of buckets

8.20 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().

8.20.1 Detailed Description

Implementation of general utility functions.

Author

Paul Griffiths

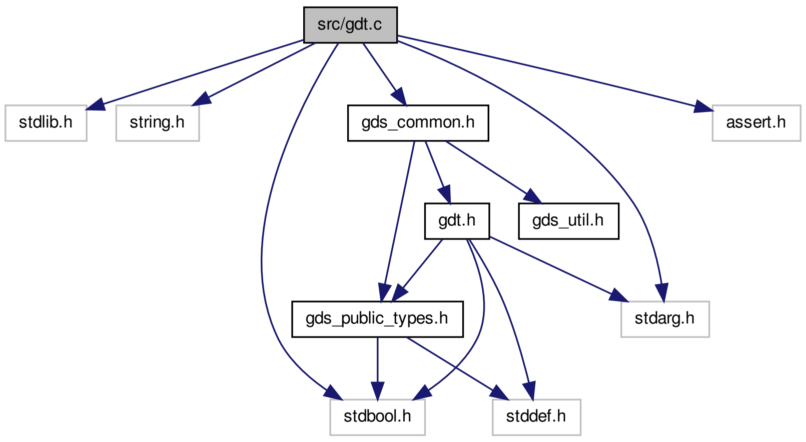
Copyright

8.21 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.

8.21.1 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/`

8.21.2 Function Documentation

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

Compare function for char.

Parameters

p1	Pointer to first value
p2	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

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

Compare function for double.

Parameters

p1	Pointer to first value
p2	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

8.21.2.3 static int gdt_compare_int (const void ∗ *p1*, const void ∗ *p2*) `[static]`

Compare function for int.

Parameters

p1	Pointer to first value
p2	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

8.21.2.4 static int gdt_compare_long (const void ∗ *p1*, const void ∗ *p2*) `[static]`

Compare function for long.

Parameters

p1	Pointer to first value
p2	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

8.21.2.5 static int gdt_compare_longlong (const void ∗ *p1*, const void ∗ *p2*) `[static]`

Compare function for long long.

Parameters

p1	Pointer to first value
p2	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

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

Compare function for signed char.

Parameters

p1	Pointer to first value
p2	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

8.21.2.7 static int gdt_compare_sizet (const void ∗ *p1*, const void ∗ *p2*) [static]

Compare function for size_t.

Parameters

p1	Pointer to first value
p2	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

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

Compare function for string.

Parameters

p1	Pointer to first value
p2	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

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

Compare function for unsigned char.

Parameters

p1	Pointer to first value
p2	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

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

Compare function for unsigned int.

Parameters

p1	Pointer to first value
p2	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

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

Compare function for unsigned long.

Parameters

p1	Pointer to first value
p2	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

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

Compare function for unsigned long long.

Parameters

p1	Pointer to first value
p2	Pointer to second value

Return values

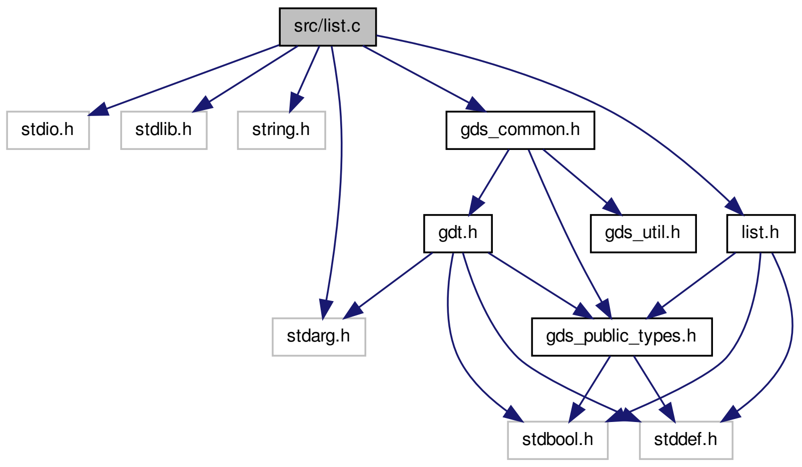
0	First value is equal to second value
-1	First value is less than second value
1	First value is greater than second value

8.22 src/list.c File Reference

Implementation of generic list data structure.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.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.
- ListItr list_find_itr (List list,...)

  Tests if a value is contained in a list.
- bool list_sort (List list)

  Sorts a list in-place, in ascending order.
- bool list_reverse_sort (List list)

  Sorts a list in-place, in descending order.
- ListItr list_itr_first (List list)

  Returns an iterator to the first element of the list.
- ListItr list_itr_last (List list)

  Returns an iterator to the last element of the list.
- ListItr list_itr_next (ListItr itr)

  Increments a list iterator.
- ListItr list_itr_previous (ListItr itr)

  Decrements a list iterator.
- void list_get_value_itr (ListItr itr, void ∗p)

  Retrieves a value from an iterator.
- bool list_is_empty (List list)

  Tests if a list is empty.
- size_t list_length (List list)

  Returns the length of a list.

8.22.1 Detailed Description

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

Author

Paul Griffiths

Copyright

8.22.2 Typedef Documentation

8.22.2.1 typedef struct list_node ∗ ListNode

List node structure

8.22.3 Function Documentation

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

Private function to insert a node into a list.

Parameters

list	A pointer to the list.
node	A pointer to the node to insert.
index	The index at which to insert.

Return values

true	Success
false	Failure, index out of range

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

Private function to return the node at a specified index.

Parameters

list	A pointer to the list.
index	The index of the requested node.

Return values

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

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

Private function to create list node.

Parameters

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

Return values

NULL	Failure, dynamic memory allocation failed
non-NULL	A pointer to the new node

8.22.3.4 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

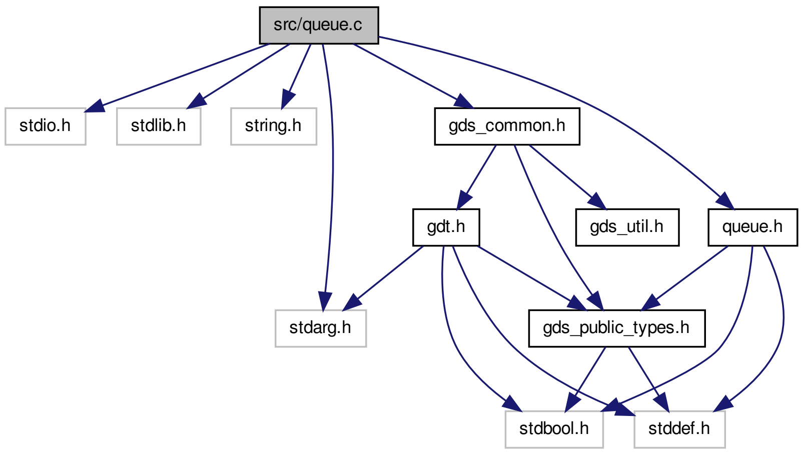
list	A pointer to the list.
node	A pointer to the node.

8.23 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.

8.23.1 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/`

8.23.2 Variable Documentation

8.23.2.1 const size_t GROWTH = 2 `[static]`

Growth factor for dynamic memory allocation.

Attention

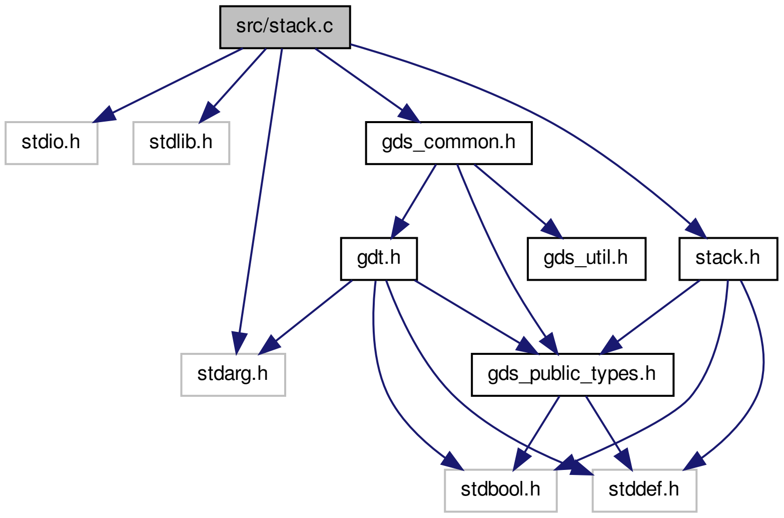
queue_push() relies on this being at least 2.

8.24 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

8.24.1 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/`

8.24.2 Variable Documentation

8.24.2.1 const size_t GROWTH = 2 [static]

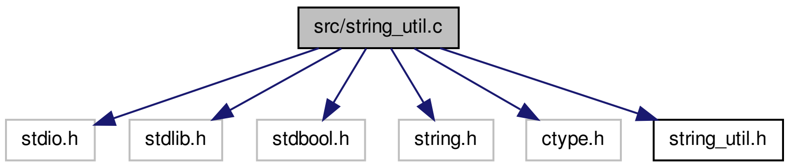
Growth factor for dynamic memory allocation

8.25 src/string_util.c File Reference

Implementation of string utility functions.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <ctype.h>
#include "string_util.h"
```

Include dependency graph for string_util.c:



Functions

- static bool list_string_resize (struct list_string ∗list, const size_t capacity)

  Helper function to resize a string list.

- char ∗ gds_trim_line_ending (char ∗str)

  Trims CR and LF characters from the end of a string.
- char ∗ gds_trim_right (char ∗str)

  Trims trailing whitespace from a string.
- char ∗ gds_trim_left (char ∗str)

  Trims leading whitespace from a string.
- char ∗ gds_trim (char ∗str)

  Trims leading and trailing whitespace from a string.
- char ∗ gds_strdup (const char ∗str)

  Dynamically duplicates a string.
- char ∗ gds_strndup (const char ∗str, const size_t n)

  Duplicates at most n characters of a string.
- struct pair_string ∗ pair_string_create (const char ∗str, const char delim)

  Splits a string into a string pair.
- struct pair_string ∗ pair_string_copy (const struct pair_string ∗pair)

  Copies a string pair.
- void pair_string_destroy (struct pair_string ∗pair)

  Destroys a string pair.
- struct list_string ∗ list_string_create (const size_t n)

  Creates a string list.
- void list_string_destroy (struct list_string ∗list)

  Destroys a string list.
- struct list_string ∗ split_string (const char ∗str, const char delim)

  Splits a string into a string list.

8.25.1 Detailed Description

Implementation of string 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/

8.25.2 Function Documentation

8.25.2.1 static bool list_string_resize (struct list_string ∗ *list*, const size_t *capacity*) [static]

Helper function to resize a string list.

Parameters

list	The string list to resize.
capacity	The new capacity.

Return values

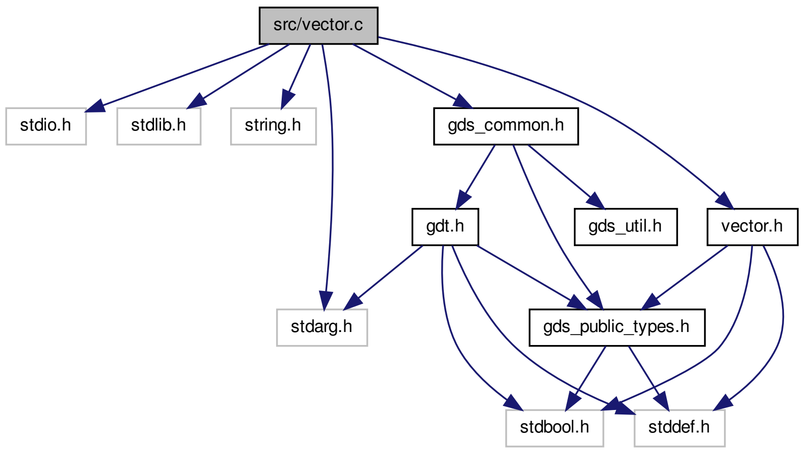
false	Failure, dynamic memory reallocation failed.
true	Success.

8.26 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

8.26.1 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/`

8.26.2 Function Documentation

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

Private function to insert a vector element.

Parameters

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

Return values

true	Success
false	Failure, dynamic reallocation failed or index out of range.

8.26.3 Variable Documentation

8.26.3.1 const size_t GROWTH = 2 `[static]`

Growth factor for dynamic memory allocation

Index

BUCKETS
    dict.c, 79
back
    queue, 51
buckets
    dict, 44

c
    gdt_generic_datatype, 45
capacity
    queue, 51
    stack, 53
    vector, 54
compfunc
    gdt_generic_datatype, 45
    list, 48
    vector, 54

d
    gdt_generic_datatype, 45
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, 45
Dict
    dict.h, 61
dict, 43
    buckets, 44
    exit_on_error, 44
    free_on_destroy, 44
    num_buckets, 44
    type, 44
dict.c
    BUCKETS, 79
    dict_buckets_create, 76
    dict_buckets_destroy, 76
    dict_create, 77
    dict_destroy, 77
    dict_has_key, 77
    dict_has_key_internal, 77
    dict_insert, 78
    dict_value_for_key, 78
    djb2hash, 78
    KVPair, 76
    kvpair_compare, 79
    kvpair_create, 79
    kvpair_destroy, 79
dict.h
    Dict, 61
    dict_create, 62
    dict_destroy, 62
    dict_has_key, 62
    dict_insert, 62
    dict_value_for_key, 63
dict_buckets_create
    dict.c, 76
dict_buckets_destroy
    dict.c, 76
dict_create
    dict.c, 77
    dict.h, 62
dict_destroy

dict.c, 77
dict.h, 62
dict_has_key
dict.c, 77
dict.h, 62
dict_has_key_internal
dict.c, 77
dict_insert
dict.c, 78
dict.h, 62
dict_value_for_key
dict.c, 78
dict.h, 63
djb2hash
dict.c, 78

element
list_node, 49
elements
queue, 51
stack, 53
vector, 54
exit_on_error
dict, 44
list, 48
queue, 51
stack, 53
vector, 54

first
pair_string, 50
free_on_destroy
dict, 44
list, 48
queue, 51
stack, 53
vector, 54
front
queue, 52

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, 90
stack.c, 92
vector.c, 96
gds.dox, 57
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
gds_error_quit
Public general generic data structures functionality, 15
gds_option
Public general generic data structures functionality, 15
gds_strdup
General purpose string manipulation functions, 32
Public general generic data structures functionality, 16
gds_strerror_quit
Public general generic data structures functionality, 16
gds_strndup
General purpose string manipulation functions, 33
gds_trim
General purpose string manipulation functions, 33
gds_trim_left
General purpose string manipulation functions, 33
gds_trim_line_ending
General purpose string manipulation functions, 33
gds_trim_right
General purpose string manipulation functions, 34
gdt.c
gdt_compare_char, 82
gdt_compare_double, 82
gdt_compare_int, 83
gdt_compare_long, 83
gdt_compare_longlong, 83
gdt_compare_schar, 83
gdt_compare_sizet, 84
gdt_compare_string, 84
gdt_compare_uchar, 84
gdt_compare_uint, 85
gdt_compare_ulong, 85
gdt_compare_ulonglong, 85
gdt_compare
Private functionality for manipulating generic datatypes, 12
gdt_compare_char
gdt.c, 82
gdt_compare_double
gdt.c, 82
gdt_compare_int
gdt.c, 83
gdt_compare_long
gdt.c, 83
gdt_compare_longlong
gdt.c, 83
gdt_compare_schar
gdt.c, 83
gdt_compare_sizet
gdt.c, 84

gdt_compare_string
    gdt.c, 84
gdt_compare_uchar
    gdt.c, 84
gdt_compare_uint
    gdt.c, 85
gdt_compare_ulong
    gdt.c, 85
gdt_compare_ulonglong
    gdt.c, 85
gdt_compare_void
    Private functionality for manipulating generic datatypes, 12
gdt_free
    Private functionality for manipulating generic datatypes, 13
gdt_generic_datatype, 44
    c, 45
    compfunc, 45
    d, 45
    data, 45
    i, 45
    l, 45
    ll, 45
    p, 45
    pc, 45
    sc, 45
    st, 45
    type, 46
    uc, 46
    ui, 46
    ul, 46
    ull, 46
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
General purpose string manipulation functions, 32
    gds_strdup, 32
    gds_strndup, 33
    gds_trim, 33
    gds_trim_left, 33
    gds_trim_line_ending, 33
    gds_trim_right, 34
    list_string_create, 34
    list_string_destroy, 34
    pair_string_copy, 34
    pair_string_create, 35
    pair_string_destroy, 35
    split_string, 35

head
    list, 48

i
    gdt_generic_datatype, 45
include/private/gds_common.h, 57
include/private/gdt.dox, 58
include/private/gdt.h, 58
include/public/dict.h, 60
include/public/gds_public_types.h, 63
include/public/gds_util.h, 64
include/public/general.dox, 65
include/public/list.dox, 65
include/public/list.h, 65
include/public/queue.dox, 68
include/public/queue.h, 68
include/public/stack.dox, 69
include/public/stack.h, 69
include/public/string_util.dox, 71
include/public/string_util.h, 71
include/public/vector.dox, 73
include/public/vector.h, 73

KVPair
    dict.c, 76
key
    kvpair, 47
kvpair, 46
    key, 47
    value, 47
kvpair_compare
    dict.c, 79
kvpair_create
    dict.c, 79
kvpair_destroy
    dict.c, 79

l
    gdt_generic_datatype, 45
length
    list, 48
    vector, 55
List
    Public interface to generic list data structure, 18
list, 47
    compfunc, 48
    exit_on_error, 48
    free_on_destroy, 48
    head, 48
    length, 48
    list_string, 50
    tail, 48
    type, 48
list.c
    list_insert_internal, 88
    list_node_at_index, 88
    list_node_create, 88
    list_node_destroy, 88
    ListNode, 88
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, 19
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, 20
list_find_itr
Public interface to generic list data structure, 20
list_get_value_itr
Public interface to generic list data structure, 20
list_insert
Public interface to generic list data structure, 20
list_insert_internal
list.c, 88
list_is_empty
Public interface to generic list data structure, 21
list_itr_first
Public interface to generic list data structure, 21
list_itr_last
Public interface to generic list data structure, 21
list_itr_next
Public interface to generic list data structure, 21
list_itr_previous
Public interface to generic list data structure, 22
list_length
Public interface to generic list data structure, 22
list_node, 49
element, 49
next, 49
prev, 49
list_node_at_index
list.c, 88
list_node_create
list.c, 88
list_node_destroy
list.c, 88
list_prepend
Public interface to generic list data structure, 22
list_reverse_sort
Public interface to generic list data structure, 22
list_set_element_at_index
Public interface to generic list data structure, 23
list_sort
Public interface to generic list data structure, 23
list_string, 49
list, 50
size, 50
list_string_create
General purpose string manipulation functions, 34
list_string_destroy
General purpose string manipulation functions, 34
list_string_resize
string_util.c, 93
ListItr
Public interface to generic list data structure, 18
ListNode
list.c, 88
ll
gdt_generic_datatype, 45

next
list_node, 49
num_buckets
dict, 44

p
gdt_generic_datatype, 45
pair_string, 50
first, 50
second, 50
pair_string_copy
General purpose string manipulation functions, 34
pair_string_create
General purpose string manipulation functions, 35
pair_string_destroy
General purpose string manipulation functions, 35
pc
gdt_generic_datatype, 45
prev
list_node, 49
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_strdup, 16
gds_strerror_quit, 16
Public interface to generic list data structure, 17
List, 18
list_append, 18
list_create, 18
list_delete_back, 18
list_delete_front, 19
list_delete_index, 19
list_destroy, 19
list_element_at_index, 19
list_find, 20
list_find_itr, 20
list_get_value_itr, 20
list_insert, 20
list_is_empty, 21
list_itr_first, 21
list_itr_last, 21
list_itr_next, 21
list_itr_previous, 22
list_length, 22
list_prepend, 22
list_reverse_sort, 22
list_set_element_at_index, 23
list_sort, 23
ListItr, 18
Public interface to generic queue data structure, 24
Queue, 24
queue_capacity, 24
queue_create, 25
queue_destroy, 25
queue_free_space, 25
queue_is_empty, 25
queue_is_full, 26
queue_peek, 26
queue_pop, 26
queue_push, 26
queue_size, 27
Public interface to generic stack data structure, 28
Stack, 28
stack_capacity, 28
stack_create, 29
stack_destroy, 29
stack_free_space, 29
stack_is_empty, 29
stack_is_full, 30
stack_peek, 30
stack_pop, 30
stack_push, 30
stack_size, 31
Public interface to generic vector data structure., 36
Vector, 37
vector_append, 37
vector_capacity, 37
vector_create, 37
vector_delete_back, 38
vector_delete_front, 38
vector_delete_index, 38
vector_destroy, 38
vector_element_at_index, 39
vector_find, 39
vector_free_space, 39
vector_insert, 39
vector_is_empty, 40
vector_length, 40
vector_prepend, 40
vector_reverse_sort, 41
vector_set_element_at_index, 41
vector_sort, 41

Queue
Public interface to generic queue data structure, 24
queue, 51
back, 51
capacity, 51
elements, 51
exit_on_error, 51
free_on_destroy, 51
front, 52
resizable, 52
size, 52
type, 52
queue.c
GROWTH, 90
queue_capacity
Public interface to generic queue data structure, 24
queue_create
Public interface to generic queue data structure, 25
queue_destroy
Public interface to generic queue data structure, 25
queue_free_space
Public interface to generic queue data structure, 25
queue_is_empty
Public interface to generic queue data structure, 25
queue_is_full
Public interface to generic queue data structure, 26
queue_peek
Public interface to generic queue data structure, 26
queue_pop
Public interface to generic queue data structure, 26
queue_push
Public interface to generic queue data structure, 26
queue_size
Public interface to generic queue data structure, 27

resizable
queue, 52
stack, 53

sc
gdt_generic_datatype, 45
second
pair_string, 50
size
list_string, 50
queue, 52
split_string

General purpose string manipulation functions, 35
src/dict.c, 75
src/gds_util.c, 80
src/gdt.c, 80
src/list.c, 86
src/queue.c, 89
src/stack.c, 90
src/string_util.c, 92
src/vector.c, 94
st
    gdt_generic_datatype, 45
Stack
    Public interface to generic stack data structure, 28
stack, 52
    capacity, 53
    elements, 53
    exit_on_error, 53
    free_on_destroy, 53
    resizable, 53
    top, 53
    type, 53
stack.c
    GROWTH, 92
stack_capacity
    Public interface to generic stack data structure, 28
stack_create
    Public interface to generic stack data structure, 29
stack_destroy
    Public interface to generic stack data structure, 29
stack_free_space
    Public interface to generic stack data structure, 29
stack_is_empty
    Public interface to generic stack data structure, 29
stack_is_full
    Public interface to generic stack data structure, 30
stack_peek
    Public interface to generic stack data structure, 30
stack_pop
    Public interface to generic stack data structure, 30
stack_push
    Public interface to generic stack data structure, 30
stack_size
    Public interface to generic stack data structure, 31
string_util.c
    list_string_resize, 93

tail
    list, 48
top
    stack, 53
type
    dict, 44
    gdt_generic_datatype, 46
    list, 48
    queue, 52
    stack, 53
    vector, 55

uc
    gdt_generic_datatype, 46
ui
    gdt_generic_datatype, 46
ul
    gdt_generic_datatype, 46
ull
    gdt_generic_datatype, 46

value
    kvpair, 47
Vector
    Public interface to generic vector data structure., 37
vector, 54
    capacity, 54
    compfunc, 54
    elements, 54
    exit_on_error, 54
    free_on_destroy, 54
    length, 55
    type, 55
vector.c
    GROWTH, 96
    vector_insert_internal, 95
vector_append
    Public interface to generic vector data structure., 37
vector_capacity
    Public interface to generic vector data structure., 37
vector_create
    Public interface to generic vector data structure., 37
vector_delete_back
    Public interface to generic vector data structure., 38
vector_delete_front
    Public interface to generic vector data structure., 38
vector_delete_index
    Public interface to generic vector data structure., 38
vector_destroy
    Public interface to generic vector data structure., 38
vector_element_at_index
    Public interface to generic vector data structure., 39
vector_find
    Public interface to generic vector data structure., 39
vector_free_space
    Public interface to generic vector data structure., 39
vector_insert
    Public interface to generic vector data structure., 39
vector_insert_internal
    vector.c, 95
vector_is_empty
    Public interface to generic vector data structure., 40
vector_length
    Public interface to generic vector data structure., 40
vector_prepend
    Public interface to generic vector data structure., 40
vector_reverse_sort
    Public interface to generic vector data structure., 41
vector_set_element_at_index
    Public interface to generic vector data structure., 41
vector_sort
    Public interface to generic vector data structure., 41