

gds

Generated by Doxygen 1.8.1.2

Fri Nov 28 2014 02:23:04

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	Public interface to string data structure	11
6.1.1	Detailed Description	12
6.1.2	Typedef Documentation	12
6.1.2.1	GDSSString	12
6.1.3	Function Documentation	13
6.1.3.1	gds_str_assign	13
6.1.3.2	gds_str_assign_cstr	13
6.1.3.3	gds_str_char_at_index	13
6.1.3.4	gds_str_clear	13
6.1.3.5	gds_str_compare	13
6.1.3.6	gds_str_compare_cstr	14
6.1.3.7	gds_str_concat	14
6.1.3.8	gds_str_concat_cstr	14
6.1.3.9	gds_str_create	14
6.1.3.10	gds_str_create_direct	15
6.1.3.11	gds_str_create_sprintf	15
6.1.3.12	gds_str_cstr	15
6.1.3.13	gds_str_decorate	16
6.1.3.14	gds_str_destroy	16

6.1.3.15	gds_str_doubleval	16
6.1.3.16	gds_str_dup	16
6.1.3.17	gds_str_getline	17
6.1.3.18	gds_str_hash	17
6.1.3.19	gds_str_intval	17
6.1.3.20	gds_str_is_alnum	17
6.1.3.21	gds_str_is_empty	18
6.1.3.22	gds_str_length	18
6.1.3.23	gds_str_size_to_fit	18
6.1.3.24	gds_str_split	18
6.1.3.25	gds_str_strchr	19
6.1.3.26	gds_str_substr_left	19
6.1.3.27	gds_str_substr_right	19
6.1.3.28	gds_str_trim	19
6.1.3.29	gds_str_trim_leading	20
6.1.3.30	gds_str_trim_trailing	20
6.1.3.31	gds_str_trunc	20
6.1.3.32	GDString_destructor	20
6.2	Public general generic data structures functionality	21
6.2.1	Detailed Description	21
6.2.2	Enumeration Type Documentation	21
6.2.2.1	gds_option	21
6.2.3	Function Documentation	21
6.2.3.1	gds_assert_quit	21
6.2.3.2	gds_error_quit	22
6.2.3.3	gds_strdup	22
6.2.3.4	gds_strerror_quit	22
6.3	Public interface to generic list data structure	23
6.3.1	Detailed Description	24
6.3.2	Typedef Documentation	24
6.3.2.1	List	24
6.3.2.2	ListItr	24
6.3.3	Function Documentation	24
6.3.3.1	list_append	24
6.3.3.2	list_create	24
6.3.3.3	list_delete_back	25
6.3.3.4	list_delete_front	25
6.3.3.5	list_delete_index	25
6.3.3.6	list_destroy	25
6.3.3.7	list_element_at_index	25

6.3.3.8	list_find	26
6.3.3.9	list_find_itr	26
6.3.3.10	list_get_value_itr	26
6.3.3.11	list_insert	27
6.3.3.12	list_is_empty	27
6.3.3.13	list_itr_first	27
6.3.3.14	list_itr_last	27
6.3.3.15	list_itr_next	28
6.3.3.16	list_itr_previous	28
6.3.3.17	list_length	28
6.3.3.18	list_prepend	28
6.3.3.19	list_reverse_sort	29
6.3.3.20	list_set_element_at_index	29
6.3.3.21	list_sort	29
6.4	Public interface to generic queue data structure	30
6.4.1	Detailed Description	30
6.4.2	Typedef Documentation	30
6.4.2.1	Queue	30
6.4.3	Function Documentation	30
6.4.3.1	queue_capacity	30
6.4.3.2	queue_create	31
6.4.3.3	queue_destroy	31
6.4.3.4	queue_free_space	31
6.4.3.5	queue_is_empty	31
6.4.3.6	queue_is_full	32
6.4.3.7	queue_peek	32
6.4.3.8	queue_pop	32
6.4.3.9	queue_push	33
6.4.3.10	queue_size	33
6.5	Public interface to generic stack data structure	34
6.5.1	Detailed Description	34
6.5.2	Typedef Documentation	34
6.5.2.1	Stack	34
6.5.3	Function Documentation	34
6.5.3.1	stack_capacity	34
6.5.3.2	stack_create	35
6.5.3.3	stack_destroy	35
6.5.3.4	stack_free_space	35
6.5.3.5	stack_is_empty	35
6.5.3.6	stack_is_full	36

6.5.3.7	stack_peek	36
6.5.3.8	stack_pop	36
6.5.3.9	stack_push	37
6.5.3.10	stack_size	37
6.6	General purpose string manipulation functions	38
6.6.1	Detailed Description	38
6.6.2	Function Documentation	38
6.6.2.1	gds_strdup	38
6.6.2.2	gds_strndup	39
6.6.2.3	gds_trim	39
6.6.2.4	gds_trim_left	39
6.6.2.5	gds_trim_line_ending	40
6.6.2.6	gds_trim_right	40
6.6.2.7	list_string_create	40
6.6.2.8	list_string_destroy	40
6.6.2.9	pair_string_copy	40
6.6.2.10	pair_string_create	41
6.6.2.11	pair_string_destroy	41
6.6.2.12	split_string	41
6.7	Public interface to generic vector data structure.	42
6.7.1	Detailed Description	42
6.7.2	Typedef Documentation	43
6.7.2.1	Vector	43
6.7.3	Function Documentation	43
6.7.3.1	vector_append	43
6.7.3.2	vector_capacity	43
6.7.3.3	vector_create	43
6.7.3.4	vector_delete_back	44
6.7.3.5	vector_delete_front	44
6.7.3.6	vector_delete_index	44
6.7.3.7	vector_destroy	44
6.7.3.8	vector_element_at_index	45
6.7.3.9	vector_find	45
6.7.3.10	vector_free_space	45
6.7.3.11	vector_insert	46
6.7.3.12	vector_is_empty	46
6.7.3.13	vector_length	46
6.7.3.14	vector_prepend	46
6.7.3.15	vector_reverse_sort	47
6.7.3.16	vector_set_element_at_index	47

6.7.3.17	vector_sort	47
6.8	Private functionality for manipulating generic datatypes	48
6.8.1	Detailed Description	48
6.8.2	Typedef Documentation	48
6.8.2.1	gds_cfunc	48
6.8.3	Enumeration Type Documentation	49
6.8.3.1	gds_datatype	49
6.8.4	Function Documentation	49
6.8.4.1	gdt_compare	49
6.8.4.2	gdt_compare_void	49
6.8.4.3	gdt_free	50
6.8.4.4	gdt_get_value	50
6.8.4.5	gdt_reverse_compare_void	50
6.8.4.6	gdt_set_value	50
7	Data Structure Documentation	53
7.1	dict Struct Reference	53
7.1.1	Detailed Description	54
7.1.2	Field Documentation	54
7.1.2.1	buckets	54
7.1.2.2	exit_on_error	54
7.1.2.3	free_on_destroy	54
7.1.2.4	num_buckets	54
7.1.2.5	type	54
7.2	GDString Struct Reference	54
7.2.1	Detailed Description	54
7.2.2	Field Documentation	54
7.2.2.1	capacity	54
7.2.2.2	data	55
7.2.2.3	length	55
7.3	gdt_generic_datatype Struct Reference	55
7.3.1	Detailed Description	55
7.3.2	Field Documentation	55
7.3.2.1	c	55
7.3.2.2	compfunc	55
7.3.2.3	d	56
7.3.2.4	data	56
7.3.2.5	i	56
7.3.2.6	l	56
7.3.2.7	ll	56

7.3.2.8	p	56
7.3.2.9	pc	56
7.3.2.10	sc	56
7.3.2.11	st	56
7.3.2.12	type	56
7.3.2.13	uc	56
7.3.2.14	ui	56
7.3.2.15	ul	57
7.3.2.16	ull	57
7.4	kvpair Struct Reference	57
7.4.1	Detailed Description	57
7.4.2	Field Documentation	57
7.4.2.1	key	57
7.4.2.2	value	57
7.5	list Struct Reference	58
7.5.1	Detailed Description	58
7.5.2	Field Documentation	58
7.5.2.1	compfunc	58
7.5.2.2	exit_on_error	58
7.5.2.3	free_on_destroy	59
7.5.2.4	head	59
7.5.2.5	length	59
7.5.2.6	tail	59
7.5.2.7	type	59
7.6	list_node Struct Reference	59
7.6.1	Detailed Description	60
7.6.2	Field Documentation	60
7.6.2.1	element	60
7.6.2.2	next	60
7.6.2.3	prev	60
7.7	list_string Struct Reference	60
7.7.1	Detailed Description	60
7.7.2	Field Documentation	60
7.7.2.1	list	60
7.7.2.2	size	60
7.8	pair_string Struct Reference	61
7.8.1	Detailed Description	61
7.8.2	Field Documentation	61
7.8.2.1	first	61
7.8.2.2	second	61

7.9	queue Struct Reference	61
7.9.1	Detailed Description	62
7.9.2	Field Documentation	62
7.9.2.1	back	62
7.9.2.2	capacity	62
7.9.2.3	elements	62
7.9.2.4	exit_on_error	62
7.9.2.5	free_on_destroy	62
7.9.2.6	front	62
7.9.2.7	resizable	62
7.9.2.8	size	62
7.9.2.9	type	63
7.10	stack Struct Reference	63
7.10.1	Detailed Description	63
7.10.2	Field Documentation	63
7.10.2.1	capacity	63
7.10.2.2	elements	63
7.10.2.3	exit_on_error	64
7.10.2.4	free_on_destroy	64
7.10.2.5	resizable	64
7.10.2.6	top	64
7.10.2.7	type	64
7.11	vector Struct Reference	64
7.11.1	Detailed Description	65
7.11.2	Field Documentation	65
7.11.2.1	capacity	65
7.11.2.2	compfunc	65
7.11.2.3	elements	65
7.11.2.4	exit_on_error	65
7.11.2.5	free_on_destroy	65
7.11.2.6	length	65
7.11.2.7	type	65
8	File Documentation	67
8.1	docs/gds_string.dox File Reference	67
8.2	docs/general.dox File Reference	67
8.3	docs/list.dox File Reference	67
8.4	docs/queue.dox File Reference	67
8.5	docs/stack.dox File Reference	67
8.6	docs/string_util.dox File Reference	67

8.7	docs/vector.dox File Reference	67
8.8	gds.dox File Reference	67
8.9	include/private/gds_common.h File Reference	67
8.9.1	Detailed Description	68
8.10	include/private/gdt.dox File Reference	68
8.11	include/private/gdt.h File Reference	69
8.11.1	Detailed Description	70
8.12	include/public/dict.h File Reference	70
8.12.1	Detailed Description	71
8.12.2	Typedef Documentation	71
8.12.2.1	Dict	71
8.12.3	Function Documentation	72
8.12.3.1	dict_create	72
8.12.3.2	dict_destroy	72
8.12.3.3	dict_has_key	72
8.12.3.4	dict_insert	72
8.12.3.5	dict_value_for_key	73
8.13	include/public/gds_public_types.h File Reference	73
8.13.1	Detailed Description	74
8.14	include/public/gds_string.h File Reference	74
8.14.1	Detailed Description	77
8.15	include/public/gds_util.h File Reference	77
8.15.1	Detailed Description	78
8.16	include/public/list.h File Reference	78
8.16.1	Detailed Description	80
8.17	include/public/queue.h File Reference	80
8.17.1	Detailed Description	81
8.18	include/public/stack.h File Reference	81
8.18.1	Detailed Description	83
8.19	include/public/string_util.h File Reference	83
8.19.1	Detailed Description	84
8.20	include/public/vector.h File Reference	84
8.20.1	Detailed Description	86
8.21	src/dict.c File Reference	86
8.21.1	Detailed Description	88
8.21.2	Typedef Documentation	88
8.21.2.1	KVPair	88
8.21.3	Function Documentation	88
8.21.3.1	dict_buckets_create	88
8.21.3.2	dict_buckets_destroy	88

8.21.3.3	dict_create	88
8.21.3.4	dict_destroy	89
8.21.3.5	dict_has_key	89
8.21.3.6	dict_has_key_internal	89
8.21.3.7	dict_insert	90
8.21.3.8	dict_value_for_key	90
8.21.3.9	djb2hash	90
8.21.3.10	kvpair_compare	90
8.21.3.11	kvpair_create	91
8.21.3.12	kvpair_destroy	91
8.21.4	Variable Documentation	91
8.21.4.1	BUCKETS	91
8.22	src/gds_string.c File Reference	91
8.22.1	Detailed Description	94
8.22.2	Function Documentation	94
8.22.2.1	change_capacity	94
8.22.2.2	change_capacity_if_needed	94
8.22.2.3	duplicate_cstr	94
8.22.2.4	gds_str_assign_cstr_direct	95
8.22.2.5	gds_str_assign_cstr_length	95
8.22.2.6	gds_str_concat_cstr_size	95
8.22.2.7	gds_str_destructor	96
8.22.2.8	gds_str_remove_left	96
8.22.2.9	gds_str_remove_right	96
8.22.2.10	truncate_if_needed	96
8.23	src/gds_util.c File Reference	96
8.23.1	Detailed Description	97
8.24	src/gdt.c File Reference	97
8.24.1	Detailed Description	99
8.24.2	Function Documentation	99
8.24.2.1	gdt_compare_char	99
8.24.2.2	gdt_compare_double	99
8.24.2.3	gdt_compare_int	100
8.24.2.4	gdt_compare_long	100
8.24.2.5	gdt_compare_longlong	100
8.24.2.6	gdt_compare_schar	100
8.24.2.7	gdt_compare_sizet	101
8.24.2.8	gdt_compare_string	101
8.24.2.9	gdt_compare_uchar	101
8.24.2.10	gdt_compare_uint	102

8.24.2.11	gdt_compare_ulong	102
8.24.2.12	gdt_compare_ulonglong	102
8.25	src/list.c File Reference	102
8.25.1	Detailed Description	104
8.25.2	Typedef Documentation	105
8.25.2.1	ListNode	105
8.25.3	Function Documentation	105
8.25.3.1	list_insert_internal	105
8.25.3.2	list_node_at_index	105
8.25.3.3	list_node_create	105
8.25.3.4	list_node_destroy	106
8.26	src/queue.c File Reference	106
8.26.1	Detailed Description	107
8.26.2	Variable Documentation	107
8.26.2.1	GROWTH	107
8.27	src/stack.c File Reference	107
8.27.1	Detailed Description	109
8.27.2	Variable Documentation	109
8.27.2.1	GROWTH	109
8.28	src/string_util.c File Reference	109
8.28.1	Detailed Description	110
8.28.2	Function Documentation	110
8.28.2.1	list_string_resize	110
8.29	src/vector.c File Reference	111
8.29.1	Detailed Description	112
8.29.2	Function Documentation	112
8.29.2.1	vector_insert_internal	112
8.29.3	Variable Documentation	113
8.29.3.1	GROWTH	113

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:

Public interface to string data structure	11
Public general generic data structures functionality	21
Public interface to generic list data structure	23
Public interface to generic queue data structure	30
Public interface to generic stack data structure	34
General purpose string manipulation functions	38
Public interface to generic vector data structure.	42
Private functionality for manipulating generic datatypes	48

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

dict	53
GDString	54
gdt_generic_datatype	
Generic datatype structure	55
kvpair	57
list	58
list_node	59
list_string	
Structure to hold a list of strings	60
pair_string	
Structure to hold a string pair	61
queue	61
stack	63
vector	64

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	67
include/private/ gdt.h	
Interface to generic data element functionality	69
include/public/ dict.h	
Interface to generic dictionary data structure	70
include/public/ gds_public_types.h	
Common public types for generic data structures library	73
include/public/ gds_string.h	
Interface to string data structure	74
include/public/ gds_util.h	
Interface to general utility functions	77
include/public/ list.h	
Interface to generic list data structure	78
include/public/ queue.h	
Interface to generic queue data structure	80
include/public/ stack.h	
Interface to generic stack data structure	81
include/public/ string_util.h	
Interface to string utility functions	83
include/public/ vector.h	
Interface to generic vector data structure	84
src/ dict.c	
Implementation of generic dictionary data structure	86
src/ gds_string.c	
Implementation of string data structure	91
src/ gds_util.c	
Implementation of general utility functions	96
src/ gdt.c	
Implementation of generic data element functionality	97
src/ list.c	
Implementation of generic list data structure	102
src/ queue.c	
Implementation of generic queue data structure	106
src/ stack.c	
Implementation of generic stack data structure	107
src/ string_util.c	
Implementation of string utility functions	109

src/ vector.c	
Implementation of generic vector data structure	111

Chapter 6

Module Documentation

6.1 Public interface to string data structure

Typedefs

- typedef struct [GDSSString](#) * [GDSSString](#)
Opaque data type for string.

Functions

- [GDSSString gds_str_create](#) (const char *init_str)
Creates a new string from a C-style string.
- [GDSSString gds_str_dup](#) ([GDSSString](#) src)
Creates a new string from another string.
- [GDSSString gds_str_create_sprintf](#) (const char *format,...)
Creates a string with `sprintf()`-type format.
- [GDSSString gds_str_create_direct](#) (char *init_str, const size_t init_str_size)
Creates a string using allocated memory.
- void [gds_str_destroy](#) ([GDSSString](#) str)
Destroys a string and releases allocated resources.
- void [GDSSString_destructor](#) (void *str)
Destroys a string and releases allocated resources.
- [GDSSString gds_str_assign](#) ([GDSSString](#) dst, [GDSSString](#) src)
Assigns a string to another.
- [GDSSString gds_str_assign_cstr](#) ([GDSSString](#) dst, const char *src)
Assigns a C-style string to a string.
- const char * [gds_str_cstr](#) ([GDSSString](#) str)
Returns a C-style string containing the string's contents.
- size_t [gds_str_length](#) ([GDSSString](#) str)
Returns the length of a string.
- [GDSSString gds_str_size_to_fit](#) ([GDSSString](#) str)
Reduces a string's capacity to fit its length.
- [GDSSString gds_str_concat](#) ([GDSSString](#) dst, [GDSSString](#) src)
Concatenates two strings.
- [GDSSString gds_str_concat_cstr](#) ([GDSSString](#) dst, const char *src)
Concatenates a C-style string to a string.
- [GDSSString gds_str_trunc](#) ([GDSSString](#) str, const size_t length)

- Truncates a string.*

 - unsigned long `gds_str_hash` (`GDSString` str)
- Calculates a hash of a string.*

 - int `gds_str_compare` (`GDSString` s1, `GDSString` s2)
- Compares two strings.*

 - int `gds_str_compare_cstr` (`GDSString` s1, const char *s2)
- Compares a string with a C-style string.*

 - int `gds_str_strchr` (`GDSString` str, const char ch, const int start)
- Returns index of first occurrence of a character.*

 - `GDSString` `gds_str_substr_left` (`GDSString` str, const size_t numchars)
- Returns a left substring.*

 - `GDSString` `gds_str_substr_right` (`GDSString` str, const size_t numchars)
- Returns a right substring.*

 - void `gds_str_split` (`GDSString` src, `GDSString` *left, `GDSString` *right, const char sc)
- Splits a string.*

 - void `gds_str_trim_leading` (`GDSString` str)
- Trims leading whitespace in-place.*

 - void `gds_str_trim_trailing` (`GDSString` str)
- Trims trailing whitespace in-place.*

 - void `gds_str_trim` (`GDSString` str)
- Trims leading and trailing whitespace in-place.*

 - char `gds_str_char_at_index` (`GDSString` str, const size_t index)
- Returns the character at a specified index.*

 - bool `gds_str_is_empty` (`GDSString` str)
- Checks if a string is empty.*

 - bool `gds_str_is_alnum` (`GDSString` str)
- Checks if a string contains only alphanumeric characters.*

 - void `gds_str_clear` (`GDSString` str)
- Clears (empties) a string.*

 - bool `gds_str_intval` (`GDSString` str, const int base, int *value)
- Gets the integer value of a string.*

 - bool `gds_str_doubleval` (`GDSString` str, double *value)
- Gets the double value of a string.*

 - `GDSString` `gds_str_getline` (`GDSString` str, const size_t size, FILE *fp)
- Gets a line from a file and assigns it to a string.*

 - `GDSString` `gds_str_decorate` (`GDSString` str, `GDSString` left_dec, `GDSString` right_dec)
- Brackets a string with decoration strings.*

6.1.1 Detailed Description

A string is an ordered collection of characters.

6.1.2 Typedef Documentation

6.1.2.1 typedef struct `GDSString`* `GDSString`

Opaque data type for string.

6.1.3 Function Documentation

6.1.3.1 `GDSString gds_str_assign (GDSString dst, GDSString src)`

Assigns a string to another.

Parameters

<i>dst</i>	The destination string.
<i>src</i>	The source string.

Returns

dst on success, `NULL` on failure.

6.1.3.2 `GDSString gds_str_assign_cstr (GDSString dst, const char * src)`

Assigns a C-style string to a string.

Parameters

<i>dst</i>	The destination string.
<i>src</i>	The source C-style string.

Returns

dst on success, `NULL` on failure.

6.1.3.3 `char gds_str_char_at_index (GDSString str, const size_t index)`

Returns the character at a specified index.

Parameters

<i>str</i>	The string.
<i>index</i>	The specified index.

Returns

The character at the specified index.

6.1.3.4 `void gds_str_clear (GDSString str)`

Clears (empties) a string.

Parameters

<i>str</i>	The string.
------------	-------------

6.1.3.5 `int gds_str_compare (GDSString s1, GDSString s2)`

Compares two strings.

Parameters

<i>s1</i>	The first string.
<i>s2</i>	The second string.

Returns

Less than, equal to, or greater than zero if *s1* is found, respectively, to be less than, equal to, or greater than *s2*.

6.1.3.6 int gds_str_compare_cstr (GDSSString *s1*, const char * *s2*)

Compares a string with a C-style string.

Parameters

<i>s1</i>	The first string.
<i>s2</i>	The second, C-Style string.

Returns

Less than, equal to, or greater than zero if *s1* is found, respectively, to be less than, equal to, or greater than *s2*.

6.1.3.7 GDSSString gds_str_concat (GDSSString *dst*, GDSSString *src*)

Concatenates two strings.

Parameters

<i>dst</i>	The destination string.
<i>src</i>	The source strings.

Returns

The destination string, or `NULL` on failure.

6.1.3.8 GDSSString gds_str_concat_cstr (GDSSString *dst*, const char * *src*)

Concatenates a C-style string to a string.

Parameters

<i>dst</i>	The destination string.
<i>src</i>	The source strings.

Returns

The destination string, or `NULL` on failure.

6.1.3.9 GDSSString gds_str_create (const char * *init_str*)

Creates a new string from a C-style string.

Parameters

<i>init_str</i>	The C-style string.
-----------------	---------------------

Returns

The new string, or `NULL` on failure.

6.1.3.10 GDString gds_str_create_direct (char * *init_str*, const size_t *init_str_size*)

Creates a string using allocated memory.

The normal construction functions duplicate the string used to create it. In cases where allocated memory is already available (e.g. in `gds_str_create_sprintf()`) this function allows that memory to be directly assigned to the string, avoiding an unnecessary duplication.

Parameters

<i>init_str</i>	The allocated memory. IMPORTANT: If the construction of the string fails, this memory will be <code>free()</code> d.
<i>init_str_size</i>	The size of the allocated memory. IMPORTANT: The string's length is assumed to be one less than this quantity, and a call to <code>strlen()</code> is NOT performed.

Returns

The new string, or `NULL` on failure.

6.1.3.11 GDString gds_str_create_sprintf (const char * *format*, ...)

Creates a string with `sprintf()`-type format.

Parameters

<i>format</i>	The format string.
...	The subsequent arguments as specified by the format string.

Returns

The new string, or `NULL` on failure.

6.1.3.12 const char* gds_str_cstr (GDString *str*)

Returns a C-style string containing the string's contents.

Parameters

<i>str</i>	The string.
------------	-------------

Returns

The C-style string containing the string's contents. The caller should not directly modify this string.

6.1.3.13 GDSString gds_str_decorate (GDSString str, GDSString left_dec, GDSString right_dec)

Brackets a string with decoration strings.

Parameters

<i>str</i>	The string to decorate.
<i>left_dec</i>	The string to add to the left of <i>str</i> .
<i>right_dec</i>	The string to add to the right of <i>str</i> , or <code>NULL</code> to add <i>left_dec</i> to both sides.

Returns

The decorated string.

6.1.3.14 void gds_str_destroy (GDSString str)

Destroys a string and releases allocated resources.

Parameters

<i>str</i>	The string to destroy..
------------	-------------------------

6.1.3.15 bool gds_str_doubleval (GDSString str, double * value)

Gets the double value of a string.

Parameters

<i>str</i>	The string.
<i>value</i>	A pointer to the double in which to store the value. Zero is stored if the string does not contain a valid double value.

Returns

`true` on successful conversion, `false` if the string does not contain a valid double value.

6.1.3.16 GDSString gds_str_dup (GDSString src)

Creates a new string from another string.

Parameters

<i>src</i>	The other string.
------------	-------------------

Returns

The new string, or `NULL` on failure.

6.1.3.17 GDSString gds_str_getline (GDSString str, const size_t size, FILE * fp)

Gets a line from a file and assigns it to a string.

Any trailing newline character is stripped.

Parameters

<i>str</i>	The string.
<i>size</i>	The maximum number of bytes to read, including the null.
<i>fp</i>	The file pointer from which to read.

Returns

`dst`

6.1.3.18 unsigned long gds_str_hash (GDSString str)

Calculates a hash of a string.

Uses Dan Bernstein's djb2 algorithm.

Parameters

<i>str</i>	The string.
------------	-------------

Returns

The hash value

6.1.3.19 bool gds_str_intval (GDSString str, const int base, int * value)

Gets the integer value of a string.

Parameters

<i>str</i>	The string.
<i>base</i>	The base of the integer. This has the same meaning as the third argument to standard C <code>strtol()</code> .
<i>value</i>	A pointer to the integer in which to store the value. Zero is stored if the string does not contain a valid integer value.

Returns

`true` on successful conversion, `false` if the string does not contain a valid integer value.

6.1.3.20 bool gds_str_is_alnum (GDSString str)

Checks if a string contains only alphanumeric characters.

The string must contain *some* alphanumeric characters to check `true`, i.e. the string must be non-empty. Thus it can be used to check that a string does indeed contain content, and that that content is solely alphanumeric.

Parameters

<i>str</i>	The string.
------------	-------------

Returns

`true` if the string contains only alphanumeric characters, `false` otherwise.

6.1.3.21 `bool gds_str_is_empty (GDSSString str)`

Checks if a string is empty.

Parameters

<i>str</i>	The string.
------------	-------------

Returns

`true` if the string is empty, `false` otherwise.

6.1.3.22 `size_t gds_str_length (GDSSString str)`

Returns the length of a string.

Parameters

<i>str</i>	The string.
------------	-------------

Returns

The length of the string.

6.1.3.23 `GDSSString gds_str_size_to_fit (GDSSString str)`

Reduces a string's capacity to fit its length.

Parameters

<i>str</i>	The string to size.
------------	---------------------

Returns

`str`, or `NULL` on failure.

6.1.3.24 `void gds_str_split (GDSSString src, GDSSString * left, GDSSString * right, const char sc)`

Splits a string.

Parameters

<i>src</i>	The string to split.
<i>left</i>	Pointer to left substring (modified)
<i>right</i>	Pointer to right substring (modified)
<i>sc</i>	Split character.

6.1.3.25 `int gds_str_strchr (GDSSString str, const char ch, const int start)`

Returns index of first occurrence of a character.

Parameters

<i>str</i>	The string.
<i>ch</i>	The character for which to search.
<i>start</i>	The index of the string at which to start looking. Set this to non-zero to begin searching from a point other than the first character of the string.

Returns

The index of the first occurrence, or -1 if the character was not found.

6.1.3.26 `GDSSString gds_str_substr_left (GDSSString str, const size_t numchars)`

Returns a left substring.

Parameters

<i>str</i>	The string.
<i>numchars</i>	The number of left characters to return. If this is greater than the length of the string, the whole string is returned.

Returns

A new string representing the substring.

6.1.3.27 `GDSSString gds_str_substr_right (GDSSString str, const size_t numchars)`

Returns a right substring.

Parameters

<i>str</i>	The string.
<i>numchars</i>	The number of right characters to return. If this is greater than the length of the string, the whole string is returned.

Returns

A new string representing the substring.

6.1.3.28 `void gds_str_trim (GDSSString str)`

Trims leading and trailing whitespace in-place.

Parameters

<i>str</i>	The string.
------------	-------------

6.1.3.29 void gds_str_trim_leading (GDSSString *str*)

Trims leading whitespace in-place.

Parameters

<i>str</i>	The string.
------------	-------------

6.1.3.30 void gds_str_trim_trailing (GDSSString *str*)

Trims trailing whitespace in-place.

Parameters

<i>str</i>	The string.
------------	-------------

6.1.3.31 GDSSString gds_str_trunc (GDSSString *str*, const size_t *length*)

Truncates a string.

Parameters

<i>str</i>	The string.
<i>length</i>	The new length to which to truncate.

Returns

The original string, or `NULL` on failure.

6.1.3.32 void GDSSString_destructor (void * *str*)

Destroys a string and releases allocated resources.

This function calls [gds_str_destroy\(\)](#), and can be passed to a data structure expecting a destructor function with the signature `void (*)(void *)`.

Parameters

<i>str</i>	The string to destroy.
------------	------------------------

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

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

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

Prints an error message exits.

Parameters

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

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

Dynamically duplicates a string.

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

Parameters

<i>str</i>	The string to duplicate.
------------	--------------------------

Return values

<i>NULL</i>	Failure, dynamic allocation failed
<i>non-NULL</i>	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

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

6.3 Public interface to generic list data structure

Typedefs

- typedef struct `list` * `List`
Opaque list type definition.
- typedef struct `list_node` * `Listltr`
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.
- `Listltr 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.
- `Listltr list_itr_first` (`List` list)
Returns an iterator to the first element of the list.
- `Listltr list_itr_last` (`List` list)
Returns an iterator to the last element of the list.
- `Listltr list_itr_next` (`Listltr` itr)
Increments a list iterator.
- `Listltr list_itr_previous` (`Listltr` itr)
Decrements a list iterator.
- void `list_get_value_itr` (`Listltr` 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

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

Return values

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

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

Creates a new list.

Parameters

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

Return values

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

6.3.3.3 bool list_delete_back (List *list*)

Deletes the value at the back of the list.

Parameters

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

Return values

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

6.3.3.4 bool list_delete_front (List *list*)

Deletes the value at the front of the list.

Parameters

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

Return values

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

6.3.3.5 bool list_delete_index (List *list*, const size_t *index*)

Deletes the value at the specified index of the list.

Parameters

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

Return values

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

6.3.3.6 void list_destroy (List *list*)

Destroys a list.

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

Parameters

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

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

Gets the value at the specified index of the list.

Parameters

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

Return values

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

6.3.3.8 `bool list_find (List list, size_t * index, ...)`

Tests if a value is contained in a list.

Parameters

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

Return values

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

6.3.3.9 `Listltr list_find_itr (List list, ...)`

Tests if a value is contained in a list.

Parameters

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

Return values

<i>NULL</i>	The value was not found in the list
<i>non-NULL</i>	A list iterator pointing to the first occurrence of the value in the list.

6.3.3.10 `void list_get_value_itr (Listltr itr, void * p)`

Retrieves a value from an iterator.

Parameters

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

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

Inserts a value into a list.

Parameters

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

Return values

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

6.3.3.12 `bool list_is_empty (List list)`

Tests if a list is empty.

Parameters

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

Return values

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

6.3.3.13 `ListIter list_itr_first (List list)`

Returns an iterator to the first element of the list.

Parameters

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

Return values

<i>NULL</i>	Failure, list is empty
<i>non-NULL</i>	An iterator to the first element of the list

6.3.3.14 `ListIter list_itr_last (List list)`

Returns an iterator to the last element of the list.

Parameters

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

Return values

<i>NULL</i>	Failure, list is empty
<i>non-NULL</i>	An iterator to the last element of the list

6.3.3.15 Listltr list_itr_next (Listltr itr)

Increments a list iterator.

Parameters

<i>itr</i>	A pointer to the iterator.
------------	----------------------------

Return values

<i>NULL</i>	End of list, no next iterator
<i>non-NULL</i>	An iterator to the next element of the list

6.3.3.16 Listltr list_itr_previous (Listltr itr)

Decrements a list iterator.

Parameters

<i>itr</i>	A pointer to the iterator.
------------	----------------------------

Return values

<i>NULL</i>	Start of list, no previous iterator
<i>non-NULL</i>	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

<i>list</i>	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

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

Return values

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

6.3.3.19 `bool list_reverse_sort (List list)`

Sorts a list in-place, in descending order.

Parameters

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

Return values

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

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

Sets the value at the specified index of the list.

Parameters

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

Return values

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

6.3.3.21 `bool list_sort (List list)`

Sorts a list in-place, in ascending order.

Parameters

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

Return values

<i>true</i>	Success
<i>false</i>	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

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

Returns

The capacity of the queue.

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

Creates a new queue.

Parameters

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

Return values

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

6.4.3.3 void queue_destroy (Queue queue)

Destroys a queue.

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

Parameters

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

6.4.3.4 size_t queue_free_space (Queue queue)

Retrieves the free space on a queue.

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

Parameters

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

Returns

The free space on the queue.

6.4.3.5 bool queue_is_empty (Queue queue)

Checks whether a queue is empty.

Parameters

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

Return values

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

6.4.3.6 `bool queue_is_full (Queue queue)`

Checks whether a queue is full.

Parameters

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

Return values

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

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

Peeks at the top value of the queue.

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

Parameters

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

Return values

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

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

Pops a value from the queue.

Parameters

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

Return values

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

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

Pushes a value onto the queue.

Parameters

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

Return values

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

Todo Rewrite to move only the required elements

6.4.3.10 `size_t queue.size (Queue queue)`

Retrieves the current size of a queue.

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

Parameters

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

Returns

The size of the queue.

6.5 Public interface to generic stack data structure

Typedefs

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

Functions

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

6.5.1 Detailed Description

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

6.5.2 Typedef Documentation

6.5.2.1 typedef struct `stack`* `Stack`

Opaque stack type definition.

6.5.3 Function Documentation

6.5.3.1 size_t `stack_capacity` (`Stack stack`)

Retrieves the current capacity of a stack.

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

Parameters

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

Returns

The capacity of the stack.

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

Creates a new stack.

Parameters

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

Return values

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

6.5.3.3 void stack_destroy (Stack stack)

Destroys a stack.

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

Parameters

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

6.5.3.4 size_t stack_free_space (Stack stack)

Retrieves the free space on a stack.

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

Parameters

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

Returns

The free space on the stack.

6.5.3.5 bool stack_is_empty (Stack stack)

Checks whether a stack is empty.

Parameters

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

Return values

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

6.5.3.6 `bool stack_is_full (Stack stack)`

Checks whether a stack is full.

Parameters

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

Return values

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

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

Peeks at the top value of the stack.

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

Parameters

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

Return values

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

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

Pops a value from the stack.

Parameters

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

Return values

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

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

Pushes a value onto the stack.

Parameters

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

Return values

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

6.5.3.10 `size_t stack_size (Stack stack)`

Retrieves the current size of a stack.

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

Parameters

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

Returns

The size of the stack.

6.6 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

<i>str</i>	The string to duplicate.
------------	--------------------------

Return values

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

Duplicates a string.

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

Parameters

<i>str</i>	The string to duplicate.
------------	--------------------------

Return values

<i>NULL</i>	Failure, dynamic allocation failed
<i>non-NULL</i>	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

<i>str</i>	The string to duplicate.
<i>n</i>	The maximum number of characters to duplicate.

Return values

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

6.6.2.3 `char* gds_trim (char * str)`

Trims leading and trailing whitespace from a string.

Parameters

<i>str</i>	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

<i>str</i>	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

<i>str</i>	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

<i>str</i>	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

<i>n</i>	The capacity of the string list.
----------	----------------------------------

Return values

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

6.6.2.8 void list_string_destroy (struct list_string * *list*)

Destroys a string list.

Parameters

<i>list</i>	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

<i>pair</i>	The string pair to copy.
-------------	--------------------------

Return values

<i>NULL</i>	Failure, dynamic memory allocation failed
<i>non-NULL</i>	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

<i>str</i>	The string to split.
<i>delim</i>	The character on which to split.

Return values

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

6.6.2.11 `void pair_string_destroy (struct pair_string * pair)`

Destroys a string pair.

Parameters

<i>pair</i>	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

<i>str</i>	The string to split.
<i>delim</i>	The delimiter character.

Return values

<i>NULL</i>	Failure, dynamic memory allocation failed
<i>non-NULL</i>	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

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

Return values

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

6.7.3.2 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

<i>vector</i>	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

<i>capacity</i>	The initial capacity for the vector.
<i>type</i>	The datatype for the vector.
<i>opts</i>	The following options can be OR'd together:

- `GDS_FREE_ON_DESTROY` to automatically `free()` pointer members when they are deleted or when the vector is destroyed. If this option is specified, then the caller should ensure that all the elements of the vector have been initialized prior to destruction.
- `GDS_EXIT_ON_ERROR` to print a message to the standard error stream and `exit()`, rather than returning a failure status.

Parameters

...	If <code>type</code> is <code>DATATYPE_POINTER</code> , this argument should be a pointer to a comparison function. In all other cases, this argument is not required, and will be ignored if it is provided.
-----	---

Return values

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

6.7.3.4 `bool vector_delete_back (Vector vector)`

Deletes the value at the back of the vector.

Parameters

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

Return values

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

6.7.3.5 `bool vector_delete_front (Vector vector)`

Deletes the value at the front of the vector.

Parameters

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

Return values

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

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

Deletes the value at the specified index of the vector.

Parameters

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

Return values

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

6.7.3.7 `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

<i>vector</i>	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

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

Return values

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

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

Tests if a value is contained in a vector.

Parameters

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

Return values

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

6.7.3.10 `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

<i>vector</i>	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

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

Return values

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

6.7.3.12 `bool vector_is_empty (Vector vector)`

Tests if a vector is empty.

Parameters

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

Return values

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

6.7.3.13 `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

<i>vector</i>	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

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

Return values

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

6.7.3.15 void vector_reverse_sort (Vector vector)

Sorts a vector in-place, in descending order.

Parameters

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

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

Sets the value at the specified index of the vector.

Parameters

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

Return values

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

6.7.3.17 void vector_sort (Vector vector)

Sorts a vector in-place, in ascending order.

Parameters

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

6.8 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.

6.8.1 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.

6.8.2 Typedef Documentation

6.8.2.1 typedef int(* gds_cfunc)(const void *, const void *)

Type definition for comparison function pointer.

6.8.3 Enumeration Type Documentation

6.8.3.1 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 *

6.8.4 Function Documentation

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

Compares two generic datatypes.

Parameters

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

Return values

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.

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

Compares two generic datatypes via void pointers.

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

Parameters

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

Return values

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

6.8.4.3 void gdt_free (struct gdt_generic_datatype * data)

Frees memory pointed to by a generic datatype.

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

Parameters

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

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

Gets the value of a generic datatype.

Parameters

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

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

Reverse compares two generic datatypes via `void` pointers.

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

Parameters

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

Return values

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

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

Sets the value of a generic datatype.

Parameters

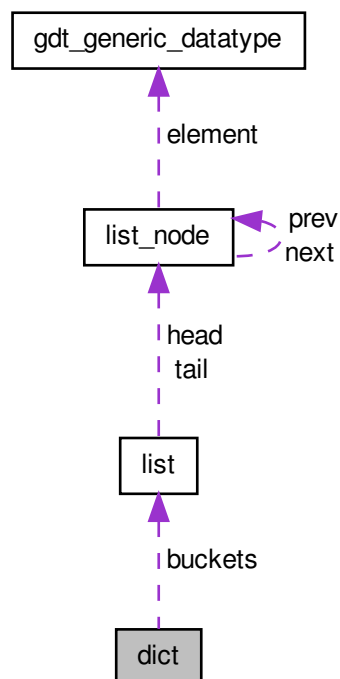
<i>data</i>	A pointer to the generic datatype.
<i>type</i>	The type of data for the datatype to contain.
<i>cfunc</i>	A pointer to a comparison function. This is ignored for all types other than <code>DATATYPE_POINTER</code> . For <code>DATATYPE_POINTER</code> , this should contain the address of a function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

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 GDSSString Struct Reference

Data Fields

- char * [data](#)
- size_t [length](#)
- size_t [capacity](#)

7.2.1 Detailed Description

Structure to contain string

7.2.2 Field Documentation

7.2.2.1 size_t GDSSString::capacity

The size of the `data` buffer

7.2.2.2 char* GDString::data

The data in C-style string format

7.2.2.3 size_t GDString::length

The length of the string

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

- [src/gds_string.c](#)

7.3 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](#)

7.3.1 Detailed Description

Generic datatype structure.

7.3.2 Field Documentation

7.3.2.1 char gdt_generic_datatype::c

char

7.3.2.2 gds_cfunc gdt_generic_datatype::compfunc

Comparison function pointer

7.3.2.3 `double gdt_generic_datatype::d`

double

7.3.2.4 `union { ... } gdt_generic_datatype::data`

Data union

7.3.2.5 `int gdt_generic_datatype::i`

int

7.3.2.6 `long gdt_generic_datatype::l`

long

7.3.2.7 `long long int gdt_generic_datatype::ll`

long long

7.3.2.8 `void* gdt_generic_datatype::p`

void *

7.3.2.9 `char* gdt_generic_datatype::pc`

char *, string

7.3.2.10 `signed char gdt_generic_datatype::sc`

signed char

7.3.2.11 `size_t gdt_generic_datatype::st`

size_t

7.3.2.12 `enum gds_datatype gdt_generic_datatype::type`

Data type

7.3.2.13 `unsigned char gdt_generic_datatype::uc`

unsigned char

7.3.2.14 `unsigned int gdt_generic_datatype::ui`

unsigned int

7.3.2.15 unsigned long gdt_generic_datatype::ul

unsigned long

7.3.2.16 unsigned long long int gdt_generic_datatype::ull

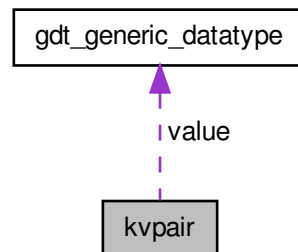
unsigned long long

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

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

7.4 kvpair Struct Reference

Collaboration diagram for kvpair:



Data Fields

- `char * key`
- `struct gdt_generic_datatype value`

7.4.1 Detailed Description

Key-Value pair structure

7.4.2 Field Documentation

7.4.2.1 `char* kvpair::key`

String key

7.4.2.2 `struct gdt_generic_datatype kvpair::value`

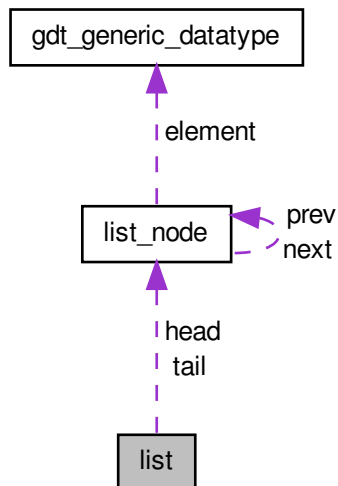
Generic datatype value

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

- [src/dict.c](#)

7.5 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](#)

7.5.1 Detailed Description

List structure

7.5.2 Field Documentation

7.5.2.1 `gds_cfunc` `list::compfunc`

Element comparison function

7.5.2.2 `bool` `list::exit_on_error`

Exit on error if true

7.5.2.3 bool list::free_on_destroy

Free pointer elements on destroy if true

7.5.2.4 struct list_node* list::head

Pointer to head of list

7.5.2.5 size_t list::length

Length of list

7.5.2.6 struct list_node* list::tail

Pointer to tail of list

7.5.2.7 enum gds_datatype list::type

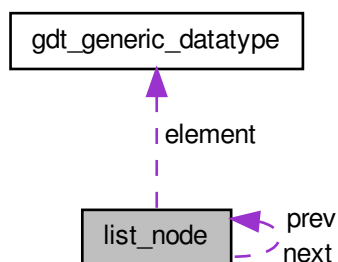
List datatype

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

- [src/list.c](#)

7.6 list_node Struct Reference

Collaboration diagram for list_node:



Data Fields

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

7.6.1 Detailed Description

List node structure

7.6.2 Field Documentation

7.6.2.1 struct gdt_generic_datatype list_node::element

Data element

7.6.2.2 struct list_node* list_node::next

Pointer to next node

7.6.2.3 struct list_node* list_node::prev

Pointer to previous node

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

- [src/list.c](#)

7.7 list_string Struct Reference

Structure to hold a list of strings.

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

Data Fields

- [size_t size](#)
- [char ** list](#)

7.7.1 Detailed Description

Structure to hold a list of strings.

7.7.2 Field Documentation

7.7.2.1 char** list_string::list

Pointer to the list

7.7.2.2 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](#)

7.8 pair_string Struct Reference

Structure to hold a string pair.

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

Data Fields

- char * [first](#)
- char * [second](#)

7.8.1 Detailed Description

Structure to hold a string pair.

7.8.2 Field Documentation

7.8.2.1 char* pair_string::first

First string of pair

7.8.2.2 char* pair_string::second

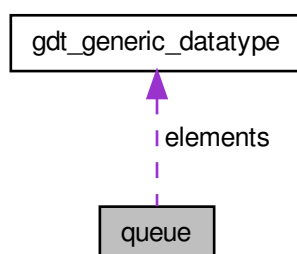
Second string of pair

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

- include/public/[string_util.h](#)

7.9 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`

7.9.1 Detailed Description

Queue structure

7.9.2 Field Documentation

7.9.2.1 `size_t queue::back`

Back of queue

7.9.2.2 `size_t queue::capacity`

Capacity of queue

7.9.2.3 `struct gdt_generic_datatype* queue::elements`

Pointer to elements

7.9.2.4 `bool queue::exit_on_error`

Exit on error if true

7.9.2.5 `bool queue::free_on_destroy`

Free pointer elements on destroy if true

7.9.2.6 `size_t queue::front`

Front of queue

7.9.2.7 `bool queue::resizable`

Dynamically resizable if true

7.9.2.8 `size_t queue::size`

Size of queue

7.9.2.9 enum gds_datatype queue::type

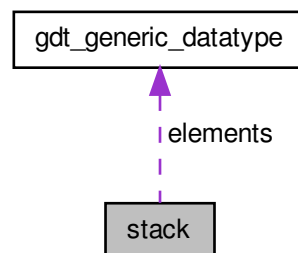
Queue datatype

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

- [src/queue.c](#)

7.10 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`

7.10.1 Detailed Description

Stack structure

7.10.2 Field Documentation

7.10.2.1 `size_t stack::capacity`

Stack capacity

7.10.2.2 `struct gdt_generic_datatype* stack::elements`

Pointer to elements

7.10.2.3 `bool stack::exit_on_error`

Exit on error if true

7.10.2.4 `bool stack::free_on_destroy`

Free pointer elements on destroy if true

7.10.2.5 `bool stack::resizable`

Dynamically resizable if true

7.10.2.6 `size_t stack::top`

Top of stack

7.10.2.7 `enum gds_datatype stack::type`

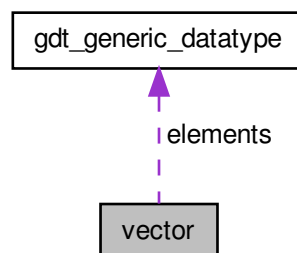
Stack datatype

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

- [src/stack.c](#)

7.11 `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](#)

7.11.1 Detailed Description

Vector structure

7.11.2 Field Documentation

7.11.2.1 `size_t vector::capacity`

Vector capacity

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

Compare function

7.11.2.3 `struct gdt_generic_datatype* vector::elements`

Pointer to elements

7.11.2.4 `bool vector::exit_on_error`

Exit on error if true

7.11.2.5 `bool vector::free_on_destroy`

Free pointer elements on destroy if true

7.11.2.6 `size_t vector::length`

Vector length

7.11.2.7 `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 docs/gds_string.dox File Reference

8.2 docs/general.dox File Reference

8.3 docs/list.dox File Reference

8.4 docs/queue.dox File Reference

8.5 docs/stack.dox File Reference

8.6 docs/string_util.dox File Reference

8.7 docs/vector.dox File Reference

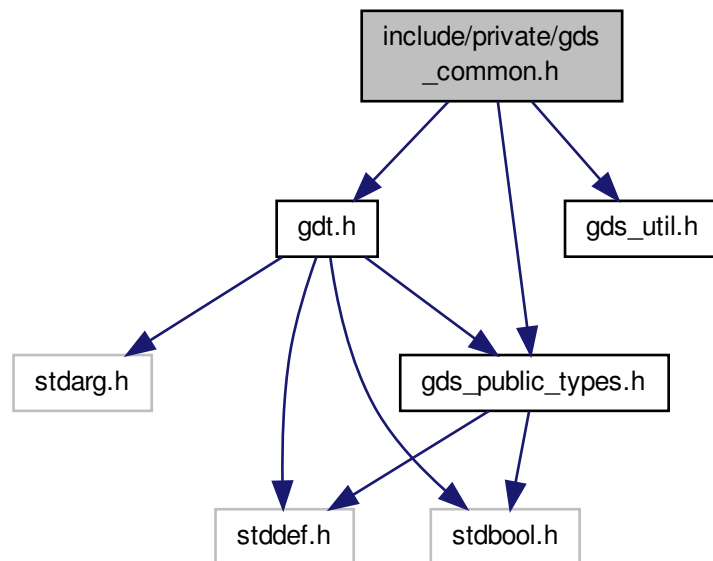
8.8 gds.dox File Reference

8.9 include/private/gds_common.h File Reference

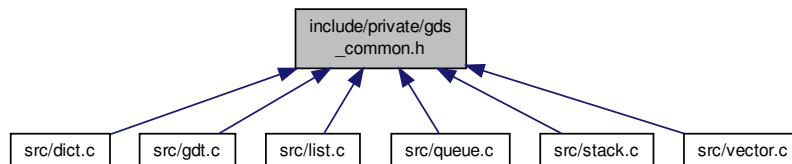
Common internal headers for data structures.

```
#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:



8.9.1 Detailed Description

Common internal headers for data structures.

Author

Paul Griffiths

Copyright

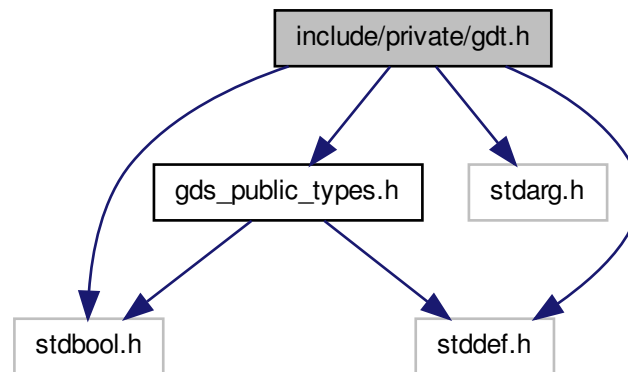
Copyright 2014 Paul Griffiths. Distributed under the terms of the GNU General Public License. <http://www.gnu.org/licenses/>

8.10 include/private/gdt.dox File Reference

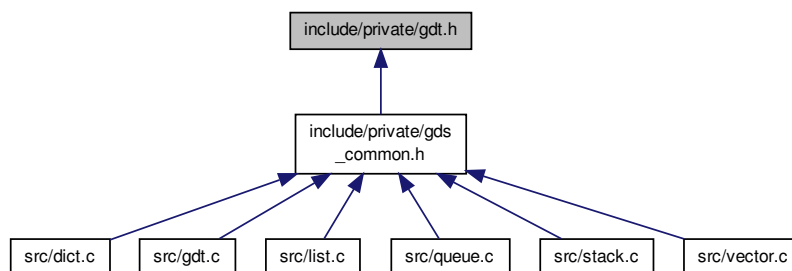
8.11 include/private/gdt.h File Reference

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.

8.11.1 Detailed Description

Interface to generic data element functionality.

Author

Paul Griffiths

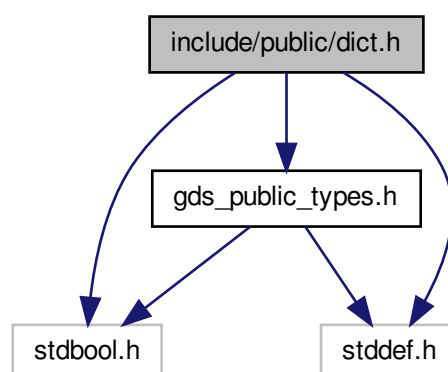
Copyright

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

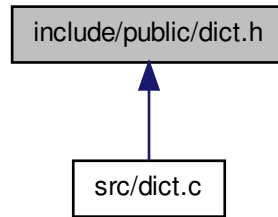
8.12 include/public/dict.h File Reference

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.

8.12.1 Detailed Description

Interface to 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/>

8.12.2 Typedef Documentation

8.12.2.1 typedef struct dict* Dict

Opaque dictionary type definition.

8.12.3 Function Documentation

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

Creates a new dictionary.

Parameters

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

Return values

<i>NULL</i>	Dictionary creation failed.
<i>non-NULL</i>	A pointer to the new dictionary.

8.12.3.2 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

<i>dict</i>	A pointer to the dictionary.
-------------	------------------------------

8.12.3.3 bool dict_has_key (Dict *dict*, const char * *key*)

Checks whether a key exists in a dictionary.

Parameters

<i>dict</i>	A pointer to the dictionary.
<i>key</i>	The key for which to search.

Return values

<i>true</i>	The key exists in the dictionary
<i>false</i>	The key does not exist in the dictionary

8.12.3.4 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

<i>dict</i>	A pointer to the dictionary.
<i>key</i>	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

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

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

Retrieves the value for a key in the dictionary.

Parameters

<i>dict</i>	A pointer to the dictionary.
<i>key</i>	The key for which to retrieve the value.
<i>p</i>	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

<i>true</i>	Success
<i>false</i>	Failure, key was not found

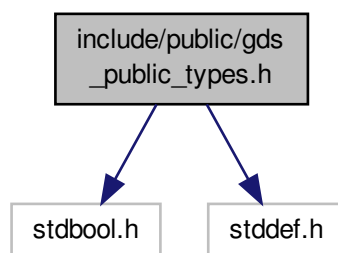
8.13 include/public/gds_public_types.h File Reference

Common public types for generic data structures library.

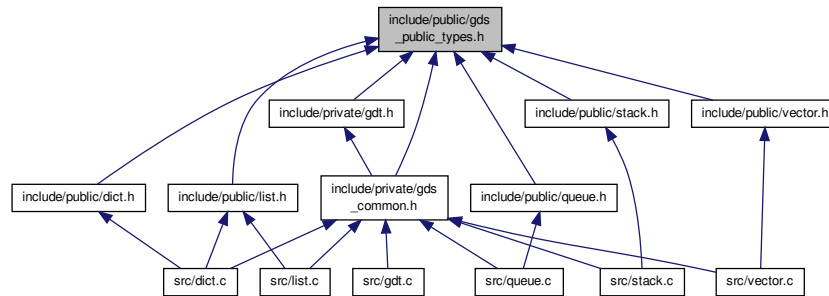
```
#include <stdbool.h>
```

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

Include dependency graph for gds_public_types.h:



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



Typedefs

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

Type definition for comparison function pointer.

Enumerations

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

Enumeration type for data structure options.

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

Enumeration type for data element type.

8.13.1 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/>

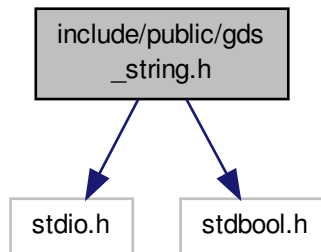
8.14 include/public/gds_string.h File Reference

Interface to string data structure.

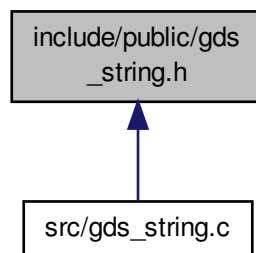
```
#include <stdio.h>
```

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

Include dependency graph for gds_string.h:



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



Typedefs

- typedef struct [GDSSString](#) * [GDSSString](#)
Opaque data type for string.

Functions

- [GDSSString gds_str_create](#) (const char *init_str)
Creates a new string from a C-style string.
- [GDSSString gds_str_dup](#) (GDSSString src)
Creates a new string from another string.
- [GDSSString gds_str_create_sprintf](#) (const char *format,...)
Creates a string with `sprintf()`-type format.
- [GDSSString gds_str_create_direct](#) (char *init_str, const size_t init_str_size)
Creates a string using allocated memory.
- void [gds_str_destroy](#) (GDSSString str)

- Destroys a string and releases allocated resources.*
- void [GDSString_destructor](#) (void *str)
- Destroys a string and releases allocated resources.*
- [GDSString gds_str_assign](#) (GDSString dst, GDSString src)
- Assigns a string to another.*
- [GDSString gds_str_assign_cstr](#) (GDSString dst, const char *src)
- Assigns a C-style string to a string.*
- const char * [gds_str_cstr](#) (GDSString str)
- Returns a C-style string containing the string's contents.*
- size_t [gds_str_length](#) (GDSString str)
- Returns the length of a string.*
- [GDSString gds_str_size_to_fit](#) (GDSString str)
- Reduces a string's capacity to fit its length.*
- [GDSString gds_str_concat](#) (GDSString dst, GDSString src)
- Concatenates two strings.*
- [GDSString gds_str_concat_cstr](#) (GDSString dst, const char *src)
- Concatenates a C-style string to a string.*
- [GDSString gds_str_trunc](#) (GDSString str, const size_t length)
- Truncates a string.*
- unsigned long [gds_str_hash](#) (GDSString str)
- Calculates a hash of a string.*
- int [gds_str_compare](#) (GDSString s1, GDSString s2)
- Compares two strings.*
- int [gds_str_compare_cstr](#) (GDSString s1, const char *s2)
- Compares a string with a C-style string.*
- int [gds_str_strchr](#) (GDSString str, const char ch, const int start)
- Returns index of first occurrence of a character.*
- [GDSString gds_str_substr_left](#) (GDSString str, const size_t numchars)
- Returns a left substring.*
- [GDSString gds_str_substr_right](#) (GDSString str, const size_t numchars)
- Returns a right substring.*
- void [gds_str_split](#) (GDSString src, GDSString *left, GDSString *right, const char sc)
- Splits a string.*
- void [gds_str_trim_leading](#) (GDSString str)
- Trims leading whitespace in-place.*
- void [gds_str_trim_trailing](#) (GDSString str)
- Trims trailing whitespace in-place.*
- void [gds_str_trim](#) (GDSString str)
- Trims leading and trailing whitespace in-place.*
- char [gds_str_char_at_index](#) (GDSString str, const size_t index)
- Returns the character at a specified index.*
- bool [gds_str_is_empty](#) (GDSString str)
- Checks if a string is empty.*
- bool [gds_str_is_alnum](#) (GDSString str)
- Checks if a string contains only alphanumeric characters.*
- void [gds_str_clear](#) (GDSString str)
- Clears (empties) a string.*
- bool [gds_str_intval](#) (GDSString str, const int base, int *value)
- Gets the integer value of a string.*
- bool [gds_str_doubleval](#) (GDSString str, double *value)
- Gets the double value of a string.*

- `GDSString gds_str_getline` (`GDSString` str, `const size_t` size, `FILE *fp`)
Gets a line from a file and assigns it to a string.
- `GDSString gds_str_decorate` (`GDSString` str, `GDSString` left_dec, `GDSString` right_dec)
Brackets a string with decoration strings.

8.14.1 Detailed Description

Interface to string data structure.

Author

Paul Griffiths

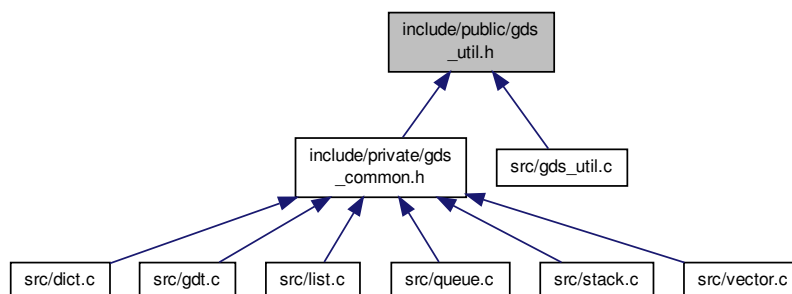
Copyright

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

8.15 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.

8.15.1 Detailed Description

Interface to general utility functions.

Author

Paul Griffiths

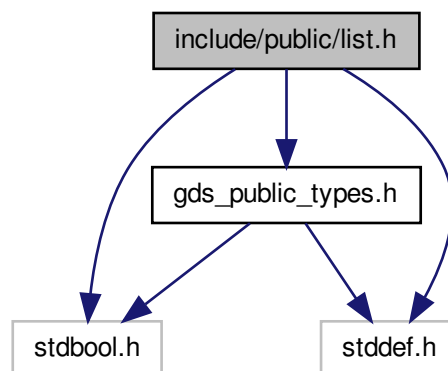
Copyright

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

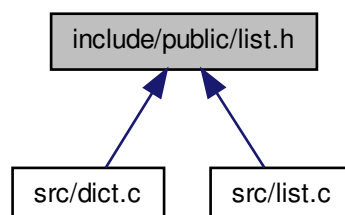
8.16 include/public/list.h File Reference

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.

8.16.1 Detailed Description

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

Author

Paul Griffiths

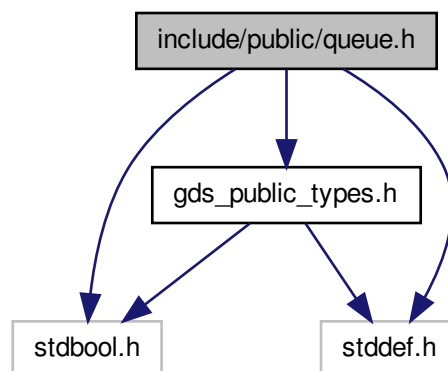
Copyright

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

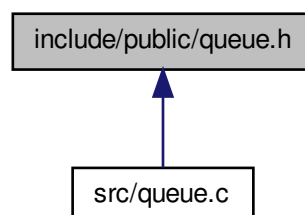
8.17 include/public/queue.h File Reference

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.

8.17.1 Detailed Description

Interface to generic queue data structure.

Author

Paul Griffiths

Copyright

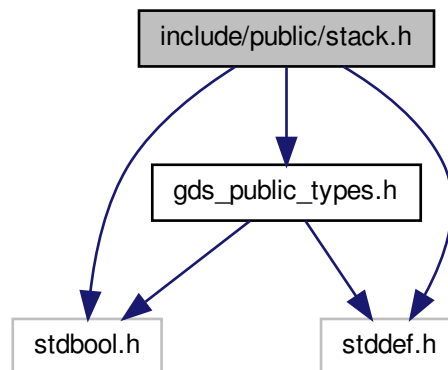
Copyright 2014 Paul Griffiths. Distributed under the terms of the GNU General Public License. <http://www.gnu.org/licenses/>

8.18 include/public/stack.h File Reference

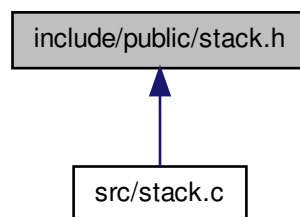
Interface to generic stack data structure.

```
#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.

8.18.1 Detailed Description

Interface to generic stack data structure.

Author

Paul Griffiths

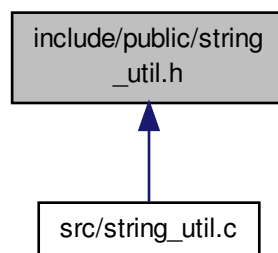
Copyright

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

8.19 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.

8.19.1 Detailed Description

Interface to string utility functions.

Author

Paul Griffiths

Copyright

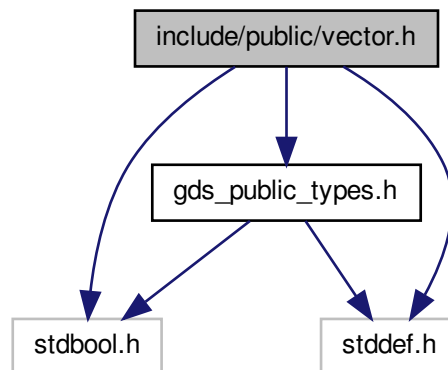
Copyright 2014 Paul Griffiths. Distributed under the terms of the GNU General Public License. <http://www.gnu.org/licenses/>

8.20 include/public/vector.h File Reference

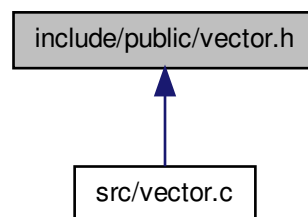
Interface to generic vector data structure.

```
#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.

8.20.1 Detailed Description

Interface to generic vector data structure.

Author

Paul Griffiths

Copyright

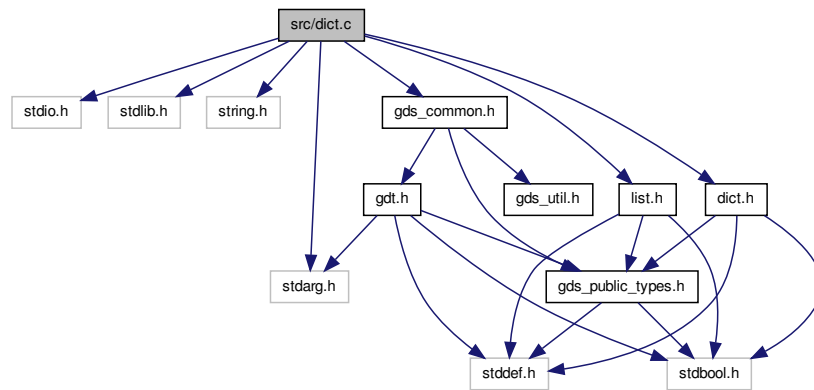
Copyright 2014 Paul Griffiths. Distributed under the terms of the GNU General Public License. <http://www.gnu.org/licenses/>

8.21 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

8.21.1 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/>

8.21.2 Typedef Documentation

8.21.2.1 typedef struct kvpair * KVPair

Key-Value pair structure

8.21.3 Function Documentation

8.21.3.1 static bool dict_buckets_create (Dict dict) [static]

Helper function to create the dictionary buckets.

Parameters

<code>dict</code>	A pointer to the dictionary.
-------------------	------------------------------

Return values

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

8.21.3.2 static void dict_buckets_destroy (Dict dict) [static]

Helper function to destroy the dictionary buckets.

Parameters

<code>dict</code>	A pointer to the dictionary.
-------------------	------------------------------

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

Creates a new dictionary.

Parameters

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

Return values

<i>NULL</i>	Dictionary creation failed.
<i>non-NULL</i>	A pointer to the new dictionary.

8.21.3.4 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

<i>dict</i>	A pointer to the dictionary.
-------------	------------------------------

8.21.3.5 bool dict_has_key (Dict dict, const char * key)

Checks whether a key exists in a dictionary.

Parameters

<i>dict</i>	A pointer to the dictionary.
<i>key</i>	The key for which to search.

Return values

<i>true</i>	The key exists in the dictionary
<i>false</i>	The key does not exist in the dictionary

8.21.3.6 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

<i>dict</i>	A pointer to the dictionary.
<i>key</i>	The key for which to search.
<i>pair</i>	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

<i>true</i>	Key was found
<i>false</i>	Key was not found

8.21.3.7 `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

<i>dict</i>	A pointer to the dictionary.
<i>key</i>	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

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

8.21.3.8 `bool dict_value_for_key (Dict dict, const char * key, void * p)`

Retrieves the value for a key in the dictionary.

Parameters

<i>dict</i>	A pointer to the dictionary.
<i>key</i>	The key for which to retrieve the value.
<i>p</i>	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

<i>true</i>	Success
<i>false</i>	Failure, key was not found

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

Calculates a hash of a string.

Uses Dan Bernstein's djb2 algorithm.

Parameters

<i>str</i>	A pointer to a string
------------	-----------------------

Returns

The hash value

8.21.3.10 `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

<i>p1</i>	A pointer to the first pair.
<i>p2</i>	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

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

Creates a new key-value pair.

Parameters

<i>key</i>	The key for the new pair.
<i>type</i>	The datatype for the new pair
<i>ap</i>	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

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

Destroys a key-value pair.

Parameters

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

8.21.4 Variable Documentation

8.21.4.1 const size_t BUCKETS = 256 [static]

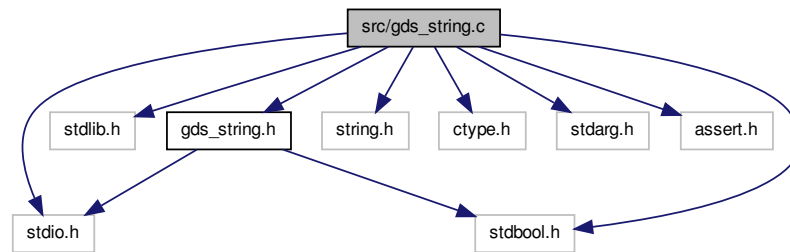
Number of buckets

8.22 src/gds_string.c File Reference

Implementation of string data structure.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <ctype.h>
#include <stdarg.h>
#include <assert.h>
#include "gds_string.h"
```

Include dependency graph for gds_string.c:



Data Structures

- struct [GDSSString](#)

Functions

- static [GDSSString gds_str_assign_cstr_direct](#) ([GDSSString](#) dst, char *src, const size_t size, const size_t length)
Directly assigns dynamically allocated data to a string.
- static [GDSSString gds_str_assign_cstr_length](#) ([GDSSString](#) dst, const char *src, const size_t length)
Assigns a C-style string to a string with length.
- static char * [duplicate_cstr](#) (const char *src, size_t *length)
Duplicates a C-style string.
- static bool [change_capacity](#) ([GDSSString](#) str, const size_t new_capacity)
Changes the capacity of a string.
- static bool [change_capacity_if_needed](#) ([GDSSString](#) str, const size_t required_capacity)
Changes the capacity of a string if needed.
- static void [truncate_if_needed](#) ([GDSSString](#) str)
Truncates a string if necessary.
- static [GDSSString gds_str_concat_cstr_size](#) ([GDSSString](#) dst, const char *src, const size_t src_length)
Concatenates a C-style string to a string, with length.
- static void [gds_str_remove_left](#) ([GDSSString](#) str, const size_t numchars)
Removes characters at the start of a string, in place.
- static void [gds_str_remove_right](#) ([GDSSString](#) str, const size_t numchars)
Removes characters at the end of a string, in place.
- [GDSSString gds_str_create_direct](#) (char *init_str, const size_t init_str_size)
Creates a string using allocated memory.
- [GDSSString gds_str_create](#) (const char *init_str)
Creates a new string from a C-style string.
- [GDSSString gds_str_dup](#) ([GDSSString](#) src)
Creates a new string from another string.
- [GDSSString gds_str_create_sprintf](#) (const char *format,...)
Creates a string with `sprintf()`-type format.
- void [gds_str_destroy](#) ([GDSSString](#) str)
Destroys a string and releases allocated resources.
- void [gds_str_destructor](#) (void *str)

- [GDSString gds_str_assign](#) (GDSString dst, GDSString src)
Assigns a string to another.
- [GDSString gds_str_assign_cstr](#) (GDSString dst, const char *src)
Assigns a C-style string to a string.
- const char * [gds_str_cstr](#) (GDSString str)
Returns a C-style string containing the string's contents.
- size_t [gds_str_length](#) (GDSString str)
Returns the length of a string.
- [GDSString gds_str_size_to_fit](#) (GDSString str)
Reduces a string's capacity to fit its length.
- [GDSString gds_str_concat](#) (GDSString dst, GDSString src)
Concatenates two strings.
- [GDSString gds_str_concat_cstr](#) (GDSString dst, const char *src)
Concatenates a C-style string to a string.
- [GDSString gds_str_trunc](#) (GDSString str, const size_t length)
Truncates a string.
- unsigned long [gds_str_hash](#) (GDSString str)
Calculates a hash of a string.
- int [gds_str_compare](#) (GDSString s1, GDSString s2)
Compares two strings.
- int [gds_str_compare_cstr](#) (GDSString s1, const char *s2)
Compares a string with a C-style string.
- int [gds_str_strchr](#) (GDSString str, const char ch, const int start)
Returns index of first occurrence of a character.
- [GDSString gds_str_substr_left](#) (GDSString str, const size_t numchars)
Returns a left substring.
- [GDSString gds_str_substr_right](#) (GDSString str, const size_t numchars)
Returns a right substring.
- void [gds_str_split](#) (GDSString src, GDSString *left, GDSString *right, const char sc)
Splits a string.
- void [gds_str_trim_leading](#) (GDSString str)
Trims leading whitespace in-place.
- void [gds_str_trim_trailing](#) (GDSString str)
Trims trailing whitespace in-place.
- void [gds_str_trim](#) (GDSString str)
Trims leading and trailing whitespace in-place.
- char [gds_str_char_at_index](#) (GDSString str, const size_t index)
Returns the character at a specified index.
- bool [gds_str_is_empty](#) (GDSString str)
Checks if a string is empty.
- bool [gds_str_is_alnum](#) (GDSString str)
Checks if a string contains only alphanumeric characters.
- void [gds_str_clear](#) (GDSString str)
Clears (empties) a string.
- bool [gds_str_intval](#) (GDSString str, const int base, int *value)
Gets the integer value of a string.
- bool [gds_str_doubleval](#) (GDSString str, double *value)
Gets the double value of a string.
- [GDSString gds_str_getline](#) (GDSString str, const size_t size, FILE *fp)
Gets a line from a file and assigns it to a string.
- [GDSString gds_str_decorate](#) (GDSString str, GDSString left_dec, GDSString right_dec)
Brackets a string with decoration strings.

8.22.1 Detailed Description

Implementation of string 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/>

8.22.2 Function Documentation

8.22.2.1 `static bool change_capacity (GDString str, const size_t new_capacity) [static]`

Changes the capacity of a string.

Parameters

<i>str</i>	The string.
<i>new_capacity</i>	The new capacity.

Returns

`true` if the capacity was successfully changed, `false` otherwise.

8.22.2.2 `static bool change_capacity_if_needed (GDString str, const size_t required_capacity) [static]`

Changes the capacity of a string if needed.

If the string's existing capacity exceeds the requirement capacity, it remains unchanged. Otherwise, the string's capacity is increased to the required capacity.

Parameters

<i>str</i>	The string.
<i>required_capacity</i>	The required capacity.

Returns

`true` if the capacity was successfully changed, or if no change was needed, `false` if a capacity change was needed but was not successful.

8.22.2.3 `static char * duplicate_cstr (const char * src, size_t length) [static]`

Duplicates a C-style string.

This can be used in place of POSIX's `strdup()`.

Parameters

<i>src</i>	The string to duplicate.
<i>length</i>	A pointer to a <code>size_t</code> variable to contain the length of the duplicated string. This is provided for efficiency purposes, as the length of the string needs to be calculated to duplicate it, so modifying this parameter may help to avoid a second unnecessary call to <code>strlen()</code> . This argument is ignored if set to <code>NULL</code> .

Returns

A pointer to the duplicated string, or `NULL` on failure. The caller is responsible for `free()` ing this string.

8.22.2.4 `static GDSString gds_str_assign_cstr_direct (GDSString dst, char * src, const size_t size, const size_t length)`
`[static]`

Directly assigns dynamically allocated data to a string.

Parameters

<i>dst</i>	The string to which to assign.
<i>src</i>	The dynamically allocated C-style string to assign.
<i>size</i>	The size of the allocated memory.
<i>length</i>	The length of the C-style string.

Returns

dst.

8.22.2.5 `static GDSString gds_str_assign_cstr_length (GDSString dst, const char * src, const size_t length)`
`[static]`

Assigns a C-style string to a string with length.

Providing the length avoids a call to `strlen()`, which is more efficient if the length is already known.

Parameters

<i>dst</i>	The string to which to assign.
<i>src</i>	The C-style string to be assigned.
<i>length</i>	The length of <i>src</i> , excluding the terminating null.

Returns

dst on success, `NULL` on failure.

8.22.2.6 `static GDSString gds_str_concat_cstr_size (GDSString dst, const char * src, const size_t src_length)`
`[static]`

Concatenates a C-style string to a string, with length.

Passing the length avoids the need to call `strlen()`, which is more efficient when we already know the length.

Parameters

<i>dst</i>	The destination string.
<i>src</i>	The C-style string to concatenate with <i>dst</i> .
<i>src_length</i>	The length of <i>src</i> , not including the terminating null.

Returns

`dst` on success, `NULL` on failure.

8.22.2.7 `void gds_str_destructor (void * str)`

8.22.2.8 `static void gds_str_remove_left (GDSSString str, const size_t numchars)` `[static]`

Removes characters at the start of a string, in place.

Parameters

<i>str</i>	The string.
<i>numchars</i>	The number of characters to remove.

8.22.2.9 `static void gds_str_remove_right (GDSSString str, const size_t numchars)` `[static]`

Removes characters at the end of a string, in place.

Parameters

<i>str</i>	The string.
<i>numchars</i>	The number of characters to remove.

8.22.2.10 `static void truncate_if_needed (GDSSString str)` `[static]`

Truncates a string if necessary.

This function truncates the length of a string, and adds a terminating null character in the last place, if the string's capacity is not sufficient to contain the string's current length. This function would normally be called after a reduction in the capacity of the string.

Parameters

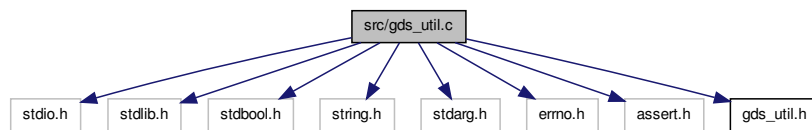
<i>str</i>	The string.
------------	-------------

8.23 `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().

8.23.1 Detailed Description

Implementation of general utility functions.

Author

Paul Griffiths

Copyright

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

8.24 src/gdt.c File Reference

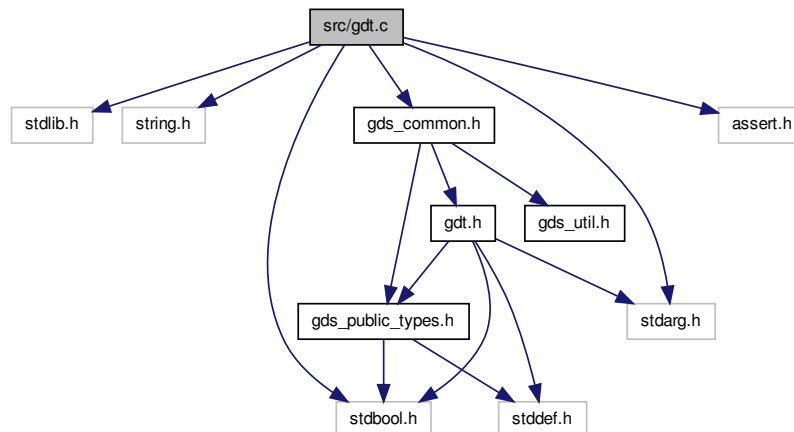
Implementation of generic data element functionality.

```

#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.

8.24.1 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/>

8.24.2 Function Documentation

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

Compare function for char.

Parameters

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

Return values

<i>0</i>	First value is equal to second value
<i>-1</i>	First value is less than second value
<i>1</i>	First value is greater than second value

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

Compare function for double.

Parameters

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

Return values

<i>0</i>	First value is equal to second value
<i>-1</i>	First value is less than second value
<i>1</i>	First value is greater than second value

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

Compare function for int.

Parameters

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

Return values

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

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

Compare function for long.

Parameters

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

Return values

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

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

Compare function for long long.

Parameters

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

Return values

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

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

Compare function for signed char.

Parameters

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

Return values

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

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

Compare function for size_t.

Parameters

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

Return values

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

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

Compare function for string.

Parameters

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

Return values

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

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

Compare function for unsigned char.

Parameters

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

Return values

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

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

Compare function for unsigned int.

Parameters

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

Return values

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

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

Compare function for unsigned long.

Parameters

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

Return values

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

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

Compare function for unsigned long long.

Parameters

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

Return values

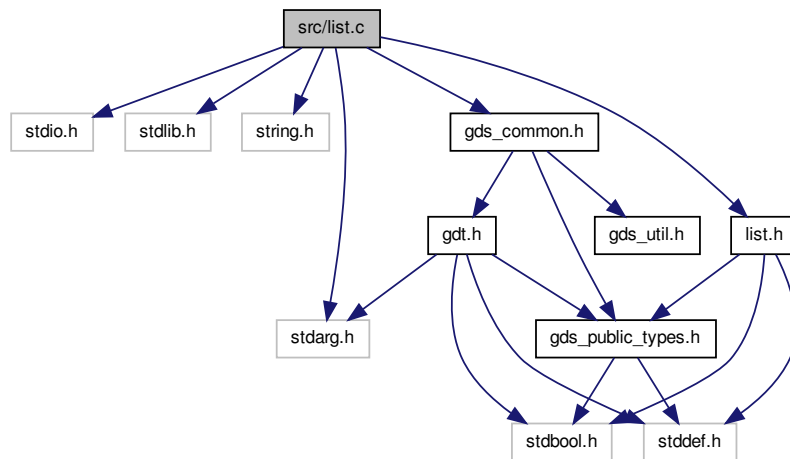
0	First value is equal to second value
-1	First value is less than second value
1	First value is greater than second value

8.25 `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.
- `Listltr 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.
- `Listltr list_itr_first (List list)`
Returns an iterator to the first element of the list.
- `Listltr list_itr_last (List list)`
Returns an iterator to the last element of the list.
- `Listltr list_itr_next (Listltr itr)`
Increments a list iterator.
- `Listltr list_itr_previous (Listltr itr)`
Decrements a list iterator.
- `void list_get_value_itr (Listltr 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.

8.25.1 Detailed Description

Implementation of 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/>

8.25.2 Typedef Documentation

8.25.2.1 typedef struct list_node * ListNode

List node structure

8.25.3 Function Documentation

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

Private function to insert a node into a list.

Parameters

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

Return values

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

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

Private function to return the node at a specified index.

Parameters

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

Return values

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

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

Private function to create list node.

Parameters

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

Return values

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

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

Destroys a list node.

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

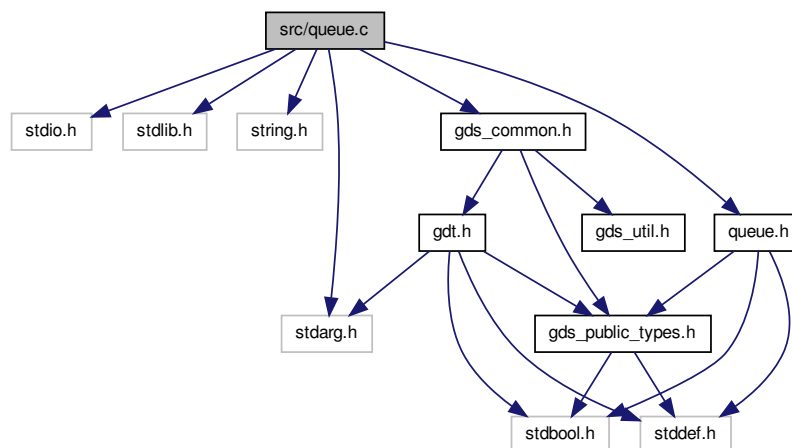
Parameters

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

8.26 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.*

8.26.1 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/>

8.26.2 Variable Documentation

8.26.2.1 `const size_t GROWTH = 2` [static]

Growth factor for dynamic memory allocation.

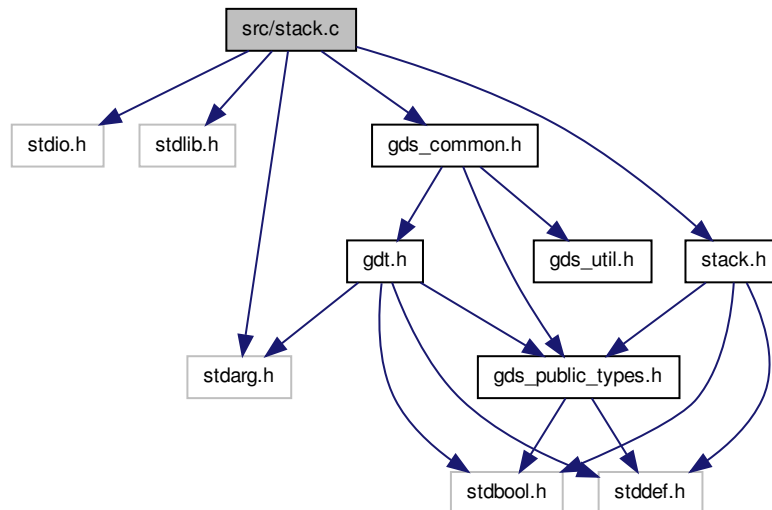
Attention

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

8.27 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`

8.27.1 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/>

8.27.2 Variable Documentation

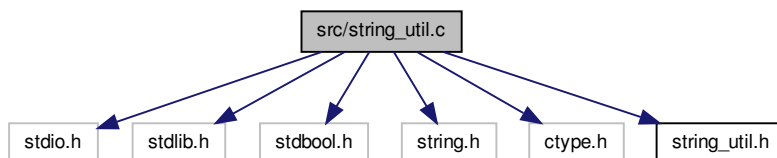
8.27.2.1 `const size_t GROWTH = 2` [static]

Growth factor for dynamic memory allocation

8.28 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.

8.28.1 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/>

8.28.2 Function Documentation

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

Helper function to resize a string list.

Parameters

<i>list</i>	The string list to resize.
<i>capacity</i>	The new capacity.

Return values

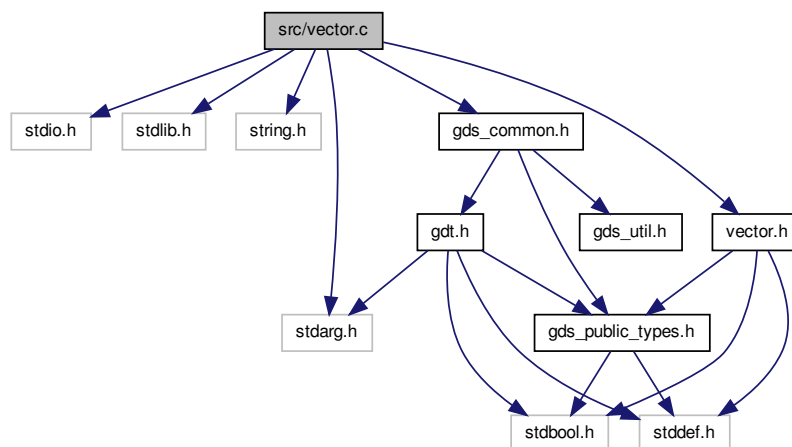
<i>false</i>	Failure, dynamic memory reallocation failed.
<i>true</i>	Success.

8.29 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

8.29.1 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/>

8.29.2 Function Documentation

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

Private function to insert a vector element.

Parameters

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

Return values

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

8.29.3 Variable Documentation

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

Growth factor for dynamic memory allocation

Index

BUCKETS
dict.c, 91

back
queue, 62

buckets
dict, 54

c
gdt_generic_datatype, 55

capacity
GDString, 54
queue, 62
stack, 63
vector, 65

change_capacity
gds_string.c, 94

change_capacity_if_needed
gds_string.c, 94

compfunc
gdt_generic_datatype, 55
list, 58
vector, 65

d
gdt_generic_datatype, 55

DATATYPE_CHAR
Private functionality for manipulating generic datatypes, 49

DATATYPE_DOUBLE
Private functionality for manipulating generic datatypes, 49

DATATYPE_INT
Private functionality for manipulating generic datatypes, 49

DATATYPE_LONG
Private functionality for manipulating generic datatypes, 49

DATATYPE_LONG_LONG
Private functionality for manipulating generic datatypes, 49

DATATYPE_POINTER
Private functionality for manipulating generic datatypes, 49

DATATYPE_SIGNED_CHAR
Private functionality for manipulating generic datatypes, 49

DATATYPE_SIZE_T
Private functionality for manipulating generic datatypes, 49

DATATYPE_STRING
Private functionality for manipulating generic datatypes, 49

DATATYPE_UNSIGNED_CHAR
Private functionality for manipulating generic datatypes, 49

DATATYPE_UNSIGNED_INT
Private functionality for manipulating generic datatypes, 49

DATATYPE_UNSIGNED_LONG
Private functionality for manipulating generic datatypes, 49

DATATYPE_UNSIGNED_LONG_LONG
Private functionality for manipulating generic datatypes, 49

data
GDString, 54
gdt_generic_datatype, 56

Dict
dict.h, 71

dict, 53
buckets, 54
exit_on_error, 54
free_on_destroy, 54
num_buckets, 54
type, 54

dict.c
BUCKETS, 91
dict_buckets_create, 88
dict_buckets_destroy, 88
dict_create, 88
dict_destroy, 89
dict_has_key, 89
dict_has_key_internal, 89
dict_insert, 89
dict_value_for_key, 90
djb2hash, 90
KVPair, 88
kvpair_compare, 90
kvpair_create, 91
kvpair_destroy, 91

dict.h
Dict, 71
dict_create, 72
dict_destroy, 72
dict_has_key, 72
dict_insert, 72
dict_value_for_key, 73

dict_buckets_create
dict.c, 88

- dict_buckets_destroy
 - dict.c, [88](#)
- dict_create
 - dict.c, [88](#)
 - dict.h, [72](#)
- dict_destroy
 - dict.c, [89](#)
 - dict.h, [72](#)
- dict_has_key
 - dict.c, [89](#)
 - dict.h, [72](#)
- dict_has_key_internal
 - dict.c, [89](#)
- dict_insert
 - dict.c, [89](#)
 - dict.h, [72](#)
- dict_value_for_key
 - dict.c, [90](#)
 - dict.h, [73](#)
- djb2hash
 - dict.c, [90](#)
- docs/gds_string.dox, [67](#)
- docs/general.dox, [67](#)
- docs/list.dox, [67](#)
- docs/queue.dox, [67](#)
- docs/stack.dox, [67](#)
- docs/string_util.dox, [67](#)
- docs/vector.dox, [67](#)
- duplicate_cstr
 - gds_string.c, [94](#)
- element
 - list_node, [60](#)
- elements
 - queue, [62](#)
 - stack, [63](#)
 - vector, [65](#)
- exit_on_error
 - dict, [54](#)
 - list, [58](#)
 - queue, [62](#)
 - stack, [63](#)
 - vector, [65](#)
- first
 - pair_string, [61](#)
- free_on_destroy
 - dict, [54](#)
 - list, [58](#)
 - queue, [62](#)
 - stack, [64](#)
 - vector, [65](#)
- front
 - queue, [62](#)
- GDS_EXIT_ON_ERROR
 - Public general generic data structures functionality, [21](#)
- GDS_FREE_ON_DESTROY
 - Public general generic data structures functionality, [21](#)
- GDS_RESIZABLE
 - Public general generic data structures functionality, [21](#)
- GDSString, [54](#)
 - capacity, [54](#)
 - data, [54](#)
 - length, [55](#)
 - Public interface to string data structure, [12](#)
- GDSString_destructor
 - Public interface to string data structure, [20](#)
- GROWTH
 - queue.c, [107](#)
 - stack.c, [109](#)
 - vector.c, [113](#)
- gds.dox, [67](#)
- gds_assert_quit
 - Public general generic data structures functionality, [21](#)
- gds_cfunc
 - Private functionality for manipulating generic datatypes, [48](#)
- gds_datatype
 - Private functionality for manipulating generic datatypes, [49](#)
- gds_error_quit
 - Public general generic data structures functionality, [21](#)
- gds_option
 - Public general generic data structures functionality, [21](#)
- gds_str_assign
 - Public interface to string data structure, [13](#)
- gds_str_assign_cstr
 - Public interface to string data structure, [13](#)
- gds_str_assign_cstr_direct
 - gds_string.c, [95](#)
- gds_str_assign_cstr_length
 - gds_string.c, [95](#)
- gds_str_char_at_index
 - Public interface to string data structure, [13](#)
- gds_str_clear
 - Public interface to string data structure, [13](#)
- gds_str_compare
 - Public interface to string data structure, [13](#)
- gds_str_compare_cstr
 - Public interface to string data structure, [14](#)
- gds_str_concat
 - Public interface to string data structure, [14](#)
- gds_str_concat_cstr
 - Public interface to string data structure, [14](#)
- gds_str_concat_cstr_size
 - gds_string.c, [95](#)
- gds_str_create
 - Public interface to string data structure, [14](#)
- gds_str_create_direct
 - Public interface to string data structure, [15](#)

- gds_str_create_sprintf
 - Public interface to string data structure, [15](#)
- gds_str_cstr
 - Public interface to string data structure, [15](#)
- gds_str_decorate
 - Public interface to string data structure, [16](#)
- gds_str_destroy
 - Public interface to string data structure, [16](#)
- gds_str_destructor
 - gds_string.c, [96](#)
- gds_str_doubleval
 - Public interface to string data structure, [16](#)
- gds_str_dup
 - Public interface to string data structure, [16](#)
- gds_str_getline
 - Public interface to string data structure, [16](#)
- gds_str_hash
 - Public interface to string data structure, [17](#)
- gds_str_intval
 - Public interface to string data structure, [17](#)
- gds_str_is_alnum
 - Public interface to string data structure, [17](#)
- gds_str_is_empty
 - Public interface to string data structure, [18](#)
- gds_str_length
 - Public interface to string data structure, [18](#)
- gds_str_remove_left
 - gds_string.c, [96](#)
- gds_str_remove_right
 - gds_string.c, [96](#)
- gds_str_size_to_fit
 - Public interface to string data structure, [18](#)
- gds_str_split
 - Public interface to string data structure, [18](#)
- gds_str_strchr
 - Public interface to string data structure, [18](#)
- gds_str_substr_left
 - Public interface to string data structure, [19](#)
- gds_str_substr_right
 - Public interface to string data structure, [19](#)
- gds_str_trim
 - Public interface to string data structure, [19](#)
- gds_str_trim_leading
 - Public interface to string data structure, [19](#)
- gds_str_trim_trailing
 - Public interface to string data structure, [20](#)
- gds_str_trunc
 - Public interface to string data structure, [20](#)
- gds_strdup
 - General purpose string manipulation functions, [38](#)
 - Public general generic data structures functionality, [22](#)
- gds_strerror_quit
 - Public general generic data structures functionality, [22](#)
- gds_string.c
 - change_capacity, [94](#)
 - change_capacity_if_needed, [94](#)
 - duplicate_cstr, [94](#)
 - gds_str_assign_cstr_direct, [95](#)
 - gds_str_assign_cstr_length, [95](#)
 - gds_str_concat_cstr_size, [95](#)
 - gds_str_destructor, [96](#)
 - gds_str_remove_left, [96](#)
 - gds_str_remove_right, [96](#)
 - truncate_if_needed, [96](#)
- gds_strndup
 - General purpose string manipulation functions, [39](#)
- gds_trim
 - General purpose string manipulation functions, [39](#)
- gds_trim_left
 - General purpose string manipulation functions, [39](#)
- gds_trim_line_ending
 - General purpose string manipulation functions, [39](#)
- gds_trim_right
 - General purpose string manipulation functions, [40](#)
- gdt.c
 - gdt_compare_char, [99](#)
 - gdt_compare_double, [99](#)
 - gdt_compare_int, [99](#)
 - gdt_compare_long, [100](#)
 - gdt_compare_longlong, [100](#)
 - gdt_compare_schar, [100](#)
 - gdt_compare_sizet, [101](#)
 - gdt_compare_string, [101](#)
 - gdt_compare_uchar, [101](#)
 - gdt_compare_uint, [101](#)
 - gdt_compare_ulong, [102](#)
 - gdt_compare_ulonglong, [102](#)
- gdt_compare
 - Private functionality for manipulating generic datatypes, [49](#)
- gdt_compare_char
 - gdt.c, [99](#)
- gdt_compare_double
 - gdt.c, [99](#)
- gdt_compare_int
 - gdt.c, [99](#)
- gdt_compare_long
 - gdt.c, [100](#)
- gdt_compare_longlong
 - gdt.c, [100](#)
- gdt_compare_schar
 - gdt.c, [100](#)
- gdt_compare_sizet
 - gdt.c, [101](#)
- gdt_compare_string
 - gdt.c, [101](#)
- gdt_compare_uchar
 - gdt.c, [101](#)
- gdt_compare_uint
 - gdt.c, [101](#)
- gdt_compare_ulong
 - gdt.c, [102](#)
- gdt_compare_ulonglong
 - gdt.c, [102](#)

- gdt_compare_void
 - Private functionality for manipulating generic datatypes, [49](#)
- gdt_free
 - Private functionality for manipulating generic datatypes, [50](#)
- gdt_generic_datatype, [55](#)
 - c, [55](#)
 - compfunc, [55](#)
 - d, [55](#)
 - data, [56](#)
 - i, [56](#)
 - l, [56](#)
 - ll, [56](#)
 - p, [56](#)
 - pc, [56](#)
 - sc, [56](#)
 - st, [56](#)
 - type, [56](#)
 - uc, [56](#)
 - ui, [56](#)
 - ul, [56](#)
 - ull, [57](#)
- gdt_get_value
 - Private functionality for manipulating generic datatypes, [50](#)
- gdt_reverse_compare_void
 - Private functionality for manipulating generic datatypes, [50](#)
- gdt_set_value
 - Private functionality for manipulating generic datatypes, [50](#)
- General purpose string manipulation functions, [38](#)
 - gds_strdup, [38](#)
 - gds_strndup, [39](#)
 - gds_trim, [39](#)
 - gds_trim_left, [39](#)
 - gds_trim_line_ending, [39](#)
 - gds_trim_right, [40](#)
 - list_string_create, [40](#)
 - list_string_destroy, [40](#)
 - pair_string_copy, [40](#)
 - pair_string_create, [41](#)
 - pair_string_destroy, [41](#)
 - split_string, [41](#)
- head
 - list, [59](#)
- i
 - gdt_generic_datatype, [56](#)
- include/private/gds_common.h, [67](#)
- include/private/gdt.dox, [68](#)
- include/private/gdt.h, [69](#)
- include/public/dict.h, [70](#)
- include/public/gds_public_types.h, [73](#)
- include/public/gds_string.h, [74](#)
- include/public/gds_util.h, [77](#)
- include/public/list.h, [78](#)
- include/public/queue.h, [80](#)
- include/public/stack.h, [81](#)
- include/public/string_util.h, [83](#)
- include/public/vector.h, [84](#)
- KVPair
 - dict.c, [88](#)
- key
 - kvpair, [57](#)
- kvpair, [57](#)
 - key, [57](#)
 - value, [57](#)
- kvpair_compare
 - dict.c, [90](#)
- kvpair_create
 - dict.c, [91](#)
- kvpair_destroy
 - dict.c, [91](#)
- l
 - gdt_generic_datatype, [56](#)
- length
 - GDSString, [55](#)
 - list, [59](#)
 - vector, [65](#)
- List
 - Public interface to generic list data structure, [24](#)
- list, [58](#)
 - compfunc, [58](#)
 - exit_on_error, [58](#)
 - free_on_destroy, [58](#)
 - head, [59](#)
 - length, [59](#)
 - list_string, [60](#)
 - tail, [59](#)
 - type, [59](#)
- list.c
 - list_insert_internal, [105](#)
 - list_node_at_index, [105](#)
 - list_node_create, [105](#)
 - list_node_destroy, [105](#)
 - ListNode, [105](#)
- list_append
 - Public interface to generic list data structure, [24](#)
- list_create
 - Public interface to generic list data structure, [24](#)
- list_delete_back
 - Public interface to generic list data structure, [24](#)
- list_delete_front
 - Public interface to generic list data structure, [25](#)
- list_delete_index
 - Public interface to generic list data structure, [25](#)
- list_destroy
 - Public interface to generic list data structure, [25](#)
- list_element_at_index
 - Public interface to generic list data structure, [25](#)
- list_find
 - Public interface to generic list data structure, [26](#)
- list_find_itr

- Public interface to generic list data structure, [26](#)
- `list_get_value_itr`
 - Public interface to generic list data structure, [26](#)
- `list_insert`
 - Public interface to generic list data structure, [26](#)
- `list_insert_internal`
 - `list.c`, [105](#)
- `list_is_empty`
 - Public interface to generic list data structure, [27](#)
- `list_itr_first`
 - Public interface to generic list data structure, [27](#)
- `list_itr_last`
 - Public interface to generic list data structure, [27](#)
- `list_itr_next`
 - Public interface to generic list data structure, [27](#)
- `list_itr_previous`
 - Public interface to generic list data structure, [28](#)
- `list_length`
 - Public interface to generic list data structure, [28](#)
- `list_node`, [59](#)
 - element, [60](#)
 - next, [60](#)
 - prev, [60](#)
- `list_node_at_index`
 - `list.c`, [105](#)
- `list_node_create`
 - `list.c`, [105](#)
- `list_node_destroy`
 - `list.c`, [105](#)
- `list_prepend`
 - Public interface to generic list data structure, [28](#)
- `list_reverse_sort`
 - Public interface to generic list data structure, [28](#)
- `list_set_element_at_index`
 - Public interface to generic list data structure, [29](#)
- `list_sort`
 - Public interface to generic list data structure, [29](#)
- `list_string`, [60](#)
 - list, [60](#)
 - size, [60](#)
- `list_string_create`
 - General purpose string manipulation functions, [40](#)
- `list_string_destroy`
 - General purpose string manipulation functions, [40](#)
- `list_string_resize`
 - `string_util.c`, [110](#)
- ListIttr
 - Public interface to generic list data structure, [24](#)
- ListNode
 - `list.c`, [105](#)
- ll
 - `gdt_generic_datatype`, [56](#)
- next
 - list_node, [60](#)
- num_buckets
 - dict, [54](#)
- p
 - `gdt_generic_datatype`, [56](#)
- pair_string, [61](#)
 - first, [61](#)
 - second, [61](#)
- pair_string_copy
 - General purpose string manipulation functions, [40](#)
- pair_string_create
 - General purpose string manipulation functions, [41](#)
- pair_string_destroy
 - General purpose string manipulation functions, [41](#)
- pc
 - `gdt_generic_datatype`, [56](#)
- prev
 - list_node, [60](#)
- Private functionality for manipulating generic datatypes, [48](#)
 - DATATYPE_CHAR, [49](#)
 - DATATYPE_DOUBLE, [49](#)
 - DATATYPE_INT, [49](#)
 - DATATYPE_LONG, [49](#)
 - DATATYPE_LONG_LONG, [49](#)
 - DATATYPE_POINTER, [49](#)
 - DATATYPE_SIGNED_CHAR, [49](#)
 - DATATYPE_SIZE_T, [49](#)
 - DATATYPE_STRING, [49](#)
 - DATATYPE_UNSIGNED_CHAR, [49](#)
 - DATATYPE_UNSIGNED_INT, [49](#)
 - DATATYPE_UNSIGNED_LONG, [49](#)
 - DATATYPE_UNSIGNED_LONG_LONG, [49](#)
 - `gds_cfunc`, [48](#)
 - `gds_datatype`, [49](#)
 - `gdt_compare`, [49](#)
 - `gdt_compare_void`, [49](#)
 - `gdt_free`, [50](#)
 - `gdt_get_value`, [50](#)
 - `gdt_reverse_compare_void`, [50](#)
 - `gdt_set_value`, [50](#)
- Public general generic data structures functionality, [21](#)
 - GDS_EXIT_ON_ERROR, [21](#)
 - GDS_FREE_ON_DESTROY, [21](#)
 - GDS_RESIZABLE, [21](#)
 - `gds_assert_quit`, [21](#)
 - `gds_error_quit`, [21](#)
 - `gds_option`, [21](#)
 - `gds_strdup`, [22](#)
 - `gds_strerror_quit`, [22](#)
- Public interface to generic list data structure, [23](#)
 - List, [24](#)
 - list_append, [24](#)
 - list_create, [24](#)
 - list_delete_back, [24](#)
 - list_delete_front, [25](#)
 - list_delete_index, [25](#)
 - list_destroy, [25](#)
 - list_element_at_index, [25](#)
 - list_find, [26](#)
 - list_find_itr, [26](#)
 - list_get_value_itr, [26](#)

- list_insert, [26](#)
- list_is_empty, [27](#)
- list_itr_first, [27](#)
- list_itr_last, [27](#)
- list_itr_next, [27](#)
- list_itr_previous, [28](#)
- list_length, [28](#)
- list_prepend, [28](#)
- list_reverse_sort, [28](#)
- list_set_element_at_index, [29](#)
- list_sort, [29](#)
- Listltr, [24](#)
- Public interface to generic queue data structure, [30](#)
 - Queue, [30](#)
 - queue_capacity, [30](#)
 - queue_create, [31](#)
 - queue_destroy, [31](#)
 - queue_free_space, [31](#)
 - queue_is_empty, [31](#)
 - queue_is_full, [32](#)
 - queue_peek, [32](#)
 - queue_pop, [32](#)
 - queue_push, [32](#)
 - queue_size, [33](#)
- Public interface to generic stack data structure, [34](#)
 - Stack, [34](#)
 - stack_capacity, [34](#)
 - stack_create, [35](#)
 - stack_destroy, [35](#)
 - stack_free_space, [35](#)
 - stack_is_empty, [35](#)
 - stack_is_full, [36](#)
 - stack_peek, [36](#)
 - stack_pop, [36](#)
 - stack_push, [36](#)
 - stack_size, [37](#)
- Public interface to generic vector data structure., [42](#)
 - Vector, [43](#)
 - vector_append, [43](#)
 - vector_capacity, [43](#)
 - vector_create, [43](#)
 - vector_delete_back, [44](#)
 - vector_delete_front, [44](#)
 - vector_delete_index, [44](#)
 - vector_destroy, [44](#)
 - vector_element_at_index, [45](#)
 - vector_find, [45](#)
 - vector_free_space, [45](#)
 - vector_insert, [45](#)
 - vector_is_empty, [46](#)
 - vector_length, [46](#)
 - vector_prepend, [46](#)
 - vector_reverse_sort, [47](#)
 - vector_set_element_at_index, [47](#)
 - vector_sort, [47](#)
- Public interface to string data structure, [11](#)
 - GDSString, [12](#)
 - GDSString_destructor, [20](#)
 - gds_str_assign, [13](#)
 - gds_str_assign_cstr, [13](#)
 - gds_str_char_at_index, [13](#)
 - gds_str_clear, [13](#)
 - gds_str_compare, [13](#)
 - gds_str_compare_cstr, [14](#)
 - gds_str_concat, [14](#)
 - gds_str_concat_cstr, [14](#)
 - gds_str_create, [14](#)
 - gds_str_create_direct, [15](#)
 - gds_str_create_sprintf, [15](#)
 - gds_str_cstr, [15](#)
 - gds_str_decorate, [16](#)
 - gds_str_destroy, [16](#)
 - gds_str_doubleval, [16](#)
 - gds_str_dup, [16](#)
 - gds_str_getline, [16](#)
 - gds_str_hash, [17](#)
 - gds_str_intval, [17](#)
 - gds_str_is_alnum, [17](#)
 - gds_str_is_empty, [18](#)
 - gds_str_length, [18](#)
 - gds_str_size_to_fit, [18](#)
 - gds_str_split, [18](#)
 - gds_str_strchr, [18](#)
 - gds_str_substr_left, [19](#)
 - gds_str_substr_right, [19](#)
 - gds_str_trim, [19](#)
 - gds_str_trim_leading, [19](#)
 - gds_str_trim_trailing, [20](#)
 - gds_str_trunc, [20](#)
- Queue
 - Public interface to generic queue data structure, [30](#)
- queue, [61](#)
 - back, [62](#)
 - capacity, [62](#)
 - elements, [62](#)
 - exit_on_error, [62](#)
 - free_on_destroy, [62](#)
 - front, [62](#)
 - resizable, [62](#)
 - size, [62](#)
 - type, [62](#)
- queue.c
 - GROWTH, [107](#)
- queue_capacity
 - Public interface to generic queue data structure, [30](#)
- queue_create
 - Public interface to generic queue data structure, [31](#)
- queue_destroy
 - Public interface to generic queue data structure, [31](#)
- queue_free_space
 - Public interface to generic queue data structure, [31](#)
- queue_is_empty
 - Public interface to generic queue data structure, [31](#)
- queue_is_full
 - Public interface to generic queue data structure, [32](#)
- queue_peek

- Public interface to generic queue data structure, [32](#)
- queue_pop
 - Public interface to generic queue data structure, [32](#)
- queue_push
 - Public interface to generic queue data structure, [32](#)
- queue_size
 - Public interface to generic queue data structure, [33](#)
- resizable
 - queue, [62](#)
 - stack, [64](#)
- sc
 - gdt_generic_datatype, [56](#)
- second
 - pair_string, [61](#)
- size
 - list_string, [60](#)
 - queue, [62](#)
- split_string
 - General purpose string manipulation functions, [41](#)
- src/dict.c, [86](#)
- src/gds_string.c, [91](#)
- src/gds_util.c, [96](#)
- src/gdt.c, [97](#)
- src/list.c, [102](#)
- src/queue.c, [106](#)
- src/stack.c, [107](#)
- src/string_util.c, [109](#)
- src/vector.c, [111](#)
- st
 - gdt_generic_datatype, [56](#)
- Stack
 - Public interface to generic stack data structure, [34](#)
- stack, [63](#)
 - capacity, [63](#)
 - elements, [63](#)
 - exit_on_error, [63](#)
 - free_on_destroy, [64](#)
 - resizable, [64](#)
 - top, [64](#)
 - type, [64](#)
- stack.c
 - GROWTH, [109](#)
- stack_capacity
 - Public interface to generic stack data structure, [34](#)
- stack_create
 - Public interface to generic stack data structure, [35](#)
- stack_destroy
 - Public interface to generic stack data structure, [35](#)
- stack_free_space
 - Public interface to generic stack data structure, [35](#)
- stack_is_empty
 - Public interface to generic stack data structure, [35](#)
- stack_is_full
 - Public interface to generic stack data structure, [36](#)
- stack_peek
 - Public interface to generic stack data structure, [36](#)
- stack_pop
 - Public interface to generic stack data structure, [36](#)
- stack_push
 - Public interface to generic stack data structure, [36](#)
- stack_size
 - Public interface to generic stack data structure, [37](#)
- string_util.c
 - list_string_resize, [110](#)
- tail
 - list, [59](#)
- top
 - stack, [64](#)
- truncate_if_needed
 - gds_string.c, [96](#)
- type
 - dict, [54](#)
 - gdt_generic_datatype, [56](#)
 - list, [59](#)
 - queue, [62](#)
 - stack, [64](#)
 - vector, [65](#)
- uc
 - gdt_generic_datatype, [56](#)
- ui
 - gdt_generic_datatype, [56](#)
- ul
 - gdt_generic_datatype, [56](#)
- ull
 - gdt_generic_datatype, [57](#)
- value
 - kvpair, [57](#)
- Vector
 - Public interface to generic vector data structure., [43](#)
- vector, [64](#)
 - capacity, [65](#)
 - compfunc, [65](#)
 - elements, [65](#)
 - exit_on_error, [65](#)
 - free_on_destroy, [65](#)
 - length, [65](#)
 - type, [65](#)
- vector.c
 - GROWTH, [113](#)
 - vector_insert_internal, [112](#)
- vector_append
 - Public interface to generic vector data structure., [43](#)
- vector_capacity
 - Public interface to generic vector data structure., [43](#)
- vector_create
 - Public interface to generic vector data structure., [43](#)
- vector_delete_back
 - Public interface to generic vector data structure., [44](#)
- vector_delete_front
 - Public interface to generic vector data structure., [44](#)
- vector_delete_index
 - Public interface to generic vector data structure., [44](#)
- vector_destroy

- Public interface to generic vector data structure., [44](#)
- `vector_element_at_index`
 - Public interface to generic vector data structure., [45](#)
- `vector_find`
 - Public interface to generic vector data structure., [45](#)
- `vector_free_space`
 - Public interface to generic vector data structure., [45](#)
- `vector_insert`
 - Public interface to generic vector data structure., [45](#)
- `vector_insert_internal`
 - `vector.c`, [112](#)
- `vector_is_empty`
 - Public interface to generic vector data structure., [46](#)
- `vector_length`
 - Public interface to generic vector data structure., [46](#)
- `vector_prepend`
 - Public interface to generic vector data structure., [46](#)
- `vector_reverse_sort`
 - Public interface to generic vector data structure., [47](#)
- `vector_set_element_at_index`
 - Public interface to generic vector data structure., [47](#)
- `vector_sort`
 - Public interface to generic vector data structure., [47](#)