

gds

Generated by Doxygen 1.8.1.2

Sun Nov 30 2014 19:54:04

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 command line parsing functionality	11
6.1.1	Detailed Description	11
6.1.2	Function Documentation	11
6.1.2.1	gds_free_options	11
6.1.2.2	gds_option_argument_int	11
6.1.2.3	gds_option_argument_string	12
6.1.2.4	gds_option_nonopt	12
6.1.2.5	gds_option_nonopts_number	12
6.1.2.6	gds_option_present	12
6.1.2.7	gds_option_progname	13
6.1.2.8	gds_parse_options	13
6.2	Public interface to string data structure	14
6.2.1	Detailed Description	15
6.2.2	Typedef Documentation	15
6.2.2.1	GDSSString	15
6.2.3	Function Documentation	15
6.2.3.1	gds_str_assign	15
6.2.3.2	gds_str_assign_cstr	16
6.2.3.3	gds_str_char_at_index	16

6.2.3.4	gds_str_clear	16
6.2.3.5	gds_str_compare	16
6.2.3.6	gds_str_compare_cstr	16
6.2.3.7	gds_str_concat	17
6.2.3.8	gds_str_concat_cstr	17
6.2.3.9	gds_str_create	17
6.2.3.10	gds_str_create_direct	17
6.2.3.11	gds_str_create_sprintf	18
6.2.3.12	gds_str_cstr	18
6.2.3.13	gds_str_decorate	18
6.2.3.14	gds_str_destroy	18
6.2.3.15	gds_str_doubleval	19
6.2.3.16	gds_str_dup	19
6.2.3.17	gds_str_getline	19
6.2.3.18	gds_str_getline_assign	19
6.2.3.19	gds_str_hash	20
6.2.3.20	gds_str_intval	20
6.2.3.21	gds_str_is_alnum	20
6.2.3.22	gds_str_is_empty	20
6.2.3.23	gds_str_length	21
6.2.3.24	gds_str_size_to_fit	21
6.2.3.25	gds_str_split	21
6.2.3.26	gds_str_strchr	21
6.2.3.27	gds_str_substr_left	22
6.2.3.28	gds_str_substr_right	22
6.2.3.29	gds_str_trim	22
6.2.3.30	gds_str_trim_leading	22
6.2.3.31	gds_str_trim_trailing	22
6.2.3.32	gds_str_trunc	23
6.2.3.33	GDSStrString_destructor	23
6.3	Private functionality for manipulating generic datatypes	24
6.3.1	Detailed Description	24
6.3.2	Typedef Documentation	24
6.3.2.1	gds_cfunc	24
6.3.3	Enumeration Type Documentation	25
6.3.3.1	gds_datatype	25
6.3.4	Function Documentation	25
6.3.4.1	gdt_compare	25
6.3.4.2	gdt_compare_void	25
6.3.4.3	gdt_free	26

6.3.4.4	gdt_get_value	26
6.3.4.5	gdt_reverse_compare_void	26
6.3.4.6	gdt_set_value	26
6.4	Public general generic data structures functionality	28
6.4.1	Detailed Description	28
6.4.2	Macro Definition Documentation	29
6.4.2.1	abort_error	29
6.4.2.2	gds_assert	29
6.4.2.3	log_error	29
6.4.2.4	log_strerror	29
6.4.2.5	quit_error	30
6.4.2.6	quit_strerror	30
6.4.2.7	xalloc	30
6.4.2.8	x fopen	31
6.4.2.9	xmalloc	31
6.4.2.10	xrealloc	31
6.4.2.11	xstrdup	31
6.4.3	Enumeration Type Documentation	31
6.4.3.1	gds_option	31
6.4.4	Function Documentation	31
6.4.4.1	gds_logerror_line	31
6.4.4.2	gds_strdup	32
6.5	Public interface to generic list data structure	33
6.5.1	Detailed Description	34
6.5.2	Typedef Documentation	34
6.5.2.1	List	34
6.5.2.2	ListItr	34
6.5.3	Function Documentation	34
6.5.3.1	list_append	34
6.5.3.2	list_create	34
6.5.3.3	list_delete_back	35
6.5.3.4	list_delete_front	35
6.5.3.5	list_delete_index	35
6.5.3.6	list_destroy	35
6.5.3.7	list_element_at_index	35
6.5.3.8	list_find	36
6.5.3.9	list_find_itr	36
6.5.3.10	list_get_value_itr	36
6.5.3.11	list_insert	37
6.5.3.12	list_is_empty	37

6.5.3.13	<code>list_itr_first</code>	37
6.5.3.14	<code>list_itr_last</code>	37
6.5.3.15	<code>list_itr_next</code>	38
6.5.3.16	<code>list_itr_previous</code>	38
6.5.3.17	<code>list_length</code>	38
6.5.3.18	<code>list_prepend</code>	38
6.5.3.19	<code>list_reverse_sort</code>	39
6.5.3.20	<code>list_set_element_at_index</code>	39
6.5.3.21	<code>list_sort</code>	39
6.6	Public interface to logging functionality	40
6.6.1	Detailed Description	40
6.6.2	Function Documentation	40
6.6.2.1	<code>gds_errlog</code>	40
6.6.2.2	<code>gds_logging_off</code>	40
6.6.2.3	<code>gds_logging_on</code>	40
6.7	Public interface to generic queue data structure	41
6.7.1	Detailed Description	41
6.7.2	Typedef Documentation	41
6.7.2.1	<code>Queue</code>	41
6.7.3	Function Documentation	41
6.7.3.1	<code>queue_capacity</code>	41
6.7.3.2	<code>queue_create</code>	42
6.7.3.3	<code>queue_destroy</code>	42
6.7.3.4	<code>queue_free_space</code>	42
6.7.3.5	<code>queue_is_empty</code>	42
6.7.3.6	<code>queue_is_full</code>	43
6.7.3.7	<code>queue_peek</code>	43
6.7.3.8	<code>queue_pop</code>	43
6.7.3.9	<code>queue_push</code>	44
6.7.3.10	<code>queue_size</code>	44
6.8	Public interface to generic stack data structure	45
6.8.1	Detailed Description	45
6.8.2	Typedef Documentation	45
6.8.2.1	<code>Stack</code>	45
6.8.3	Function Documentation	45
6.8.3.1	<code>stack_capacity</code>	45
6.8.3.2	<code>stack_create</code>	46
6.8.3.3	<code>stack_destroy</code>	46
6.8.3.4	<code>stack_free_space</code>	46
6.8.3.5	<code>stack_is_empty</code>	46

6.8.3.6	stack_is_full	47
6.8.3.7	stack_peek	47
6.8.3.8	stack_pop	47
6.8.3.9	stack_push	48
6.8.3.10	stack_size	48
6.9	General purpose string manipulation functions	49
6.9.1	Detailed Description	49
6.9.2	Function Documentation	49
6.9.2.1	gds_strdup	49
6.9.2.2	gds_strndup	50
6.9.2.3	gds_trim	50
6.9.2.4	gds_trim_left	50
6.9.2.5	gds_trim_line_ending	51
6.9.2.6	gds_trim_right	51
6.9.2.7	list_string_create	51
6.9.2.8	list_string_destroy	51
6.9.2.9	pair_string_copy	51
6.9.2.10	pair_string_create	52
6.9.2.11	pair_string_destroy	52
6.9.2.12	split_string	52
6.10	Public interface to unit testing functionality	53
6.10.1	Detailed Description	53
6.10.2	Macro Definition Documentation	54
6.10.2.1	RUN_CASE	54
6.10.2.2	TEST_ASSERT_ALMOST_EQUAL	54
6.10.2.3	TEST_ASSERT_EQUAL	54
6.10.2.4	TEST_ASSERT_FALSE	54
6.10.2.5	TEST_ASSERT_NOTEQUAL	55
6.10.2.6	TEST_ASSERT_STR_EQUAL	55
6.10.2.7	TEST_ASSERT_STR_NOTEQUAL	55
6.10.2.8	TEST_ASSERT_TRUE	56
6.10.2.9	TEST_CASE	56
6.10.2.10	TEST_SUITE	56
6.10.3	Function Documentation	56
6.10.3.1	tests_assert_almost_equal	56
6.10.3.2	tests_assert_true	57
6.10.3.3	tests_get_failures	57
6.10.3.4	tests_get_successes	57
6.10.3.5	tests_get_total_tests	57
6.10.3.6	tests_initialize	58

6.10.3.7	tests_report	58
6.11	Public interface to generic vector data structure.	59
6.11.1	Detailed Description	59
6.11.2	Typedef Documentation	60
6.11.2.1	Vector	60
6.11.3	Function Documentation	60
6.11.3.1	vector_append	60
6.11.3.2	vector_capacity	60
6.11.3.3	vector_create	60
6.11.3.4	vector_delete_back	61
6.11.3.5	vector_delete_front	61
6.11.3.6	vector_delete_index	61
6.11.3.7	vector_destroy	61
6.11.3.8	vector_element_at_index	62
6.11.3.9	vector_find	62
6.11.3.10	vector_free_space	62
6.11.3.11	vector_insert	63
6.11.3.12	vector_is_empty	63
6.11.3.13	vector_length	63
6.11.3.14	vector_prepend	63
6.11.3.15	vector_reverse_sort	64
6.11.3.16	vector_set_element_at_index	64
6.11.3.17	vector_sort	64
7	Data Structure Documentation	65
7.1	dict Struct Reference	65
7.1.1	Detailed Description	66
7.1.2	Field Documentation	66
7.1.2.1	buckets	66
7.1.2.2	exit_on_error	66
7.1.2.3	free_on_destroy	66
7.1.2.4	num_buckets	66
7.1.2.5	type	66
7.2	gds_kvpair Struct Reference	66
7.2.1	Detailed Description	67
7.2.2	Field Documentation	67
7.2.2.1	key	67
7.2.2.2	value	67
7.3	GDSSString Struct Reference	67
7.3.1	Detailed Description	68

7.3.2	Field Documentation	68
7.3.2.1	capacity	68
7.3.2.2	data	68
7.3.2.3	length	68
7.4	gdt_generic_datatype Struct Reference	68
7.4.1	Detailed Description	69
7.4.2	Field Documentation	69
7.4.2.1	c	69
7.4.2.2	compfunc	69
7.4.2.3	d	69
7.4.2.4	data	69
7.4.2.5	gdsstr	69
7.4.2.6	i	69
7.4.2.7	l	69
7.4.2.8	ll	70
7.4.2.9	p	70
7.4.2.10	pc	70
7.4.2.11	sc	70
7.4.2.12	st	70
7.4.2.13	type	70
7.4.2.14	uc	70
7.4.2.15	ui	70
7.4.2.16	ul	70
7.4.2.17	ull	70
7.5	list Struct Reference	71
7.5.1	Detailed Description	71
7.5.2	Field Documentation	71
7.5.2.1	compfunc	71
7.5.2.2	exit_on_error	72
7.5.2.3	free_on_destroy	72
7.5.2.4	head	72
7.5.2.5	length	72
7.5.2.6	tail	72
7.5.2.7	type	72
7.6	list_node Struct Reference	72
7.6.1	Detailed Description	73
7.6.2	Field Documentation	73
7.6.2.1	element	73
7.6.2.2	next	73
7.6.2.3	prev	73

7.7	list_string Struct Reference	73
7.7.1	Detailed Description	73
7.7.2	Field Documentation	73
7.7.2.1	list	73
7.7.2.2	size	74
7.8	pair_string Struct Reference	74
7.8.1	Detailed Description	74
7.8.2	Field Documentation	74
7.8.2.1	first	74
7.8.2.2	second	74
7.9	queue Struct Reference	75
7.9.1	Detailed Description	75
7.9.2	Field Documentation	75
7.9.2.1	back	75
7.9.2.2	capacity	75
7.9.2.3	elements	76
7.9.2.4	exit_on_error	76
7.9.2.5	free_on_destroy	76
7.9.2.6	front	76
7.9.2.7	resizable	76
7.9.2.8	size	76
7.9.2.9	type	76
7.10	stack Struct Reference	77
7.10.1	Detailed Description	77
7.10.2	Field Documentation	77
7.10.2.1	capacity	77
7.10.2.2	elements	77
7.10.2.3	exit_on_error	77
7.10.2.4	free_on_destroy	78
7.10.2.5	resizable	78
7.10.2.6	top	78
7.10.2.7	type	78
7.11	vector Struct Reference	78
7.11.1	Detailed Description	79
7.11.2	Field Documentation	79
7.11.2.1	capacity	79
7.11.2.2	compfunc	79
7.11.2.3	elements	79
7.11.2.4	exit_on_error	79
7.11.2.5	free_on_destroy	79

7.11.2.6	length	79
7.11.2.7	type	79
8	File Documentation	81
8.1	docs/cmdline.dox File Reference	81
8.2	docs/gds.dox File Reference	81
8.3	docs/gds_string.dox File Reference	81
8.4	docs/gdt.dox File Reference	81
8.5	docs/general.dox File Reference	81
8.6	docs/list.dox File Reference	81
8.7	docs/logging.dox File Reference	81
8.8	docs/queue.dox File Reference	81
8.9	docs/stack.dox File Reference	81
8.10	docs/string_util.dox File Reference	81
8.11	docs/unittest.dox File Reference	81
8.12	docs/vector.dox File Reference	81
8.13	include/private/pggds_internal/gds_common.h File Reference	81
8.13.1	Detailed Description	82
8.14	include/private/pggds_internal/gdt.h File Reference	82
8.14.1	Detailed Description	84
8.15	include/public/pggds/dict.h File Reference	84
8.15.1	Detailed Description	85
8.15.2	Typedef Documentation	85
8.15.2.1	Dict	85
8.15.3	Function Documentation	86
8.15.3.1	dict_create	86
8.15.3.2	dict_destroy	86
8.15.3.3	dict_has_key	86
8.15.3.4	dict_insert	86
8.15.3.5	dict_value_for_key	87
8.16	include/public/pggds/gds_opt.h File Reference	87
8.16.1	Detailed Description	88
8.17	include/public/pggds/gds_public_types.h File Reference	88
8.17.1	Detailed Description	89
8.18	include/public/pggds/gds_string.h File Reference	89
8.18.1	Detailed Description	92
8.19	include/public/pggds/gds_util.h File Reference	92
8.19.1	Detailed Description	93
8.20	include/public/pggds/gds_util_error.h File Reference	93
8.20.1	Detailed Description	94

8.20.2	Enumeration Type Documentation	94
8.20.2.1	gds_error_quit_type	94
8.21	include/public/pggds/gds_util_logging.h File Reference	94
8.21.1	Detailed Description	95
8.21.2	Macro Definition Documentation	96
8.21.2.1	DPRINTF	96
8.21.3	Function Documentation	96
8.21.3.1	gds_log_msg	96
8.22	include/public/pggds/gds_util_std_wrappers.h File Reference	96
8.22.1	Detailed Description	97
8.22.2	Function Documentation	97
8.22.2.1	gds_xcalloc	97
8.22.2.2	gds_xfopen	98
8.22.2.3	gds_xmalloc	98
8.22.2.4	gds_xrealloc	98
8.22.2.5	gds_xstrdup	99
8.23	include/public/pggds/gds_util_string.h File Reference	99
8.23.1	Detailed Description	99
8.24	include/public/pggds/kvpair.h File Reference	99
8.24.1	Detailed Description	101
8.24.2	Typedef Documentation	101
8.24.2.1	KVPair	101
8.24.3	Function Documentation	101
8.24.3.1	gds_kvpair_compare	101
8.24.3.2	gds_kvpair_create	101
8.24.3.3	gds_kvpair_destroy	102
8.25	include/public/pggds/list.h File Reference	102
8.25.1	Detailed Description	104
8.26	include/public/pggds/queue.h File Reference	104
8.26.1	Detailed Description	106
8.27	include/public/pggds/stack.h File Reference	106
8.27.1	Detailed Description	108
8.28	include/public/pggds/string_util.h File Reference	108
8.28.1	Detailed Description	109
8.29	include/public/pggds/test_logging.h File Reference	110
8.29.1	Detailed Description	111
8.30	include/public/pggds/unittest.h File Reference	111
8.30.1	Detailed Description	112
8.31	include/public/pggds/vector.h File Reference	112
8.31.1	Detailed Description	114

8.32	src/dict.c File Reference	114
8.32.1	Detailed Description	116
8.32.2	Function Documentation	116
8.32.2.1	dict_buckets_create	116
8.32.2.2	dict_buckets_destroy	116
8.32.2.3	dict_create	116
8.32.2.4	dict_destroy	117
8.32.2.5	dict_has_key	117
8.32.2.6	dict_has_key_internal	117
8.32.2.7	dict_insert	117
8.32.2.8	dict_value_for_key	118
8.32.2.9	djb2hash	118
8.32.3	Variable Documentation	118
8.32.3.1	BUCKETS	118
8.33	src/gds_opt.c File Reference	118
8.33.1	Detailed Description	120
8.33.2	Macro Definition Documentation	120
8.33.2.1	GDSDEBUG	120
8.33.3	Enumeration Type Documentation	120
8.33.3.1	gds_argument_type	120
8.33.4	Function Documentation	120
8.33.4.1	create_static_structures	120
8.33.4.2	destroy_static_structures	120
8.33.4.3	gds_get_recognized_options	120
8.33.5	Variable Documentation	121
8.33.5.1	nonopts	121
8.33.5.2	options	121
8.33.5.3	parsed	121
8.33.5.4	progname	121
8.34	src/gds_string.c File Reference	121
8.34.1	Detailed Description	123
8.34.2	Function Documentation	124
8.34.2.1	change_capacity	124
8.34.2.2	change_capacity_if_needed	124
8.34.2.3	duplicate_cstr	124
8.34.2.4	gds_str_assign_cstr_direct	124
8.34.2.5	gds_str_assign_cstr_length	125
8.34.2.6	gds_str_concat_cstr_size	125
8.34.2.7	gds_str_destructor	125
8.34.2.8	gds_str_remove_left	125

8.34.2.9	gds_str_remove_right	126
8.34.2.10	truncate_if_needed	126
8.35	src/gds_util_error.c File Reference	126
8.35.1	Detailed Description	126
8.36	src/gds_util_logging.c File Reference	127
8.36.1	Detailed Description	127
8.36.2	Function Documentation	128
8.36.2.1	gds_log_msg	128
8.36.3	Variable Documentation	128
8.36.3.1	gds_error_file	128
8.36.3.2	gds_error_file_name	128
8.36.3.3	gds_logging_enabled	128
8.37	src/gds_util_std_wrappers.c File Reference	128
8.37.1	Detailed Description	129
8.37.2	Function Documentation	129
8.37.2.1	gds_xcalloc	129
8.37.2.2	gds_xfopen	129
8.37.2.3	gds_xmalloc	130
8.37.2.4	gds_xrealloc	130
8.37.2.5	gds_xstrdup	130
8.38	src/gdt.c File Reference	131
8.38.1	Detailed Description	132
8.38.2	Function Documentation	132
8.38.2.1	gdt_compare_char	132
8.38.2.2	gdt_compare_double	132
8.38.2.3	gdt_compare_gds_str	133
8.38.2.4	gdt_compare_int	133
8.38.2.5	gdt_compare_long	133
8.38.2.6	gdt_compare_longlong	134
8.38.2.7	gdt_compare_schar	134
8.38.2.8	gdt_compare_sizet	134
8.38.2.9	gdt_compare_string	134
8.38.2.10	gdt_compare_uchar	135
8.38.2.11	gdt_compare_uint	135
8.38.2.12	gdt_compare_ulong	135
8.38.2.13	gdt_compare_ulonglong	136
8.39	src/kvpair.c File Reference	136
8.39.1	Detailed Description	136
8.39.2	Function Documentation	137
8.39.2.1	gds_kvpair_compare	137

8.39.2.2	gds_kvpair_create	137
8.39.2.3	gds_kvpair_destroy	137
8.40	src/list.c File Reference	138
8.40.1	Detailed Description	139
8.40.2	Typedef Documentation	139
8.40.2.1	ListNode	139
8.40.3	Function Documentation	140
8.40.3.1	list_insert_internal	140
8.40.3.2	list_node_at_index	140
8.40.3.3	list_node_create	140
8.40.3.4	list_node_destroy	140
8.41	src/queue.c File Reference	141
8.41.1	Detailed Description	142
8.41.2	Variable Documentation	142
8.41.2.1	GROWTH	142
8.42	src/stack.c File Reference	142
8.42.1	Detailed Description	143
8.42.2	Variable Documentation	143
8.42.2.1	GROWTH	143
8.43	src/string_util.c File Reference	144
8.43.1	Detailed Description	145
8.43.2	Function Documentation	145
8.43.2.1	list_string_resize	145
8.44	src/test_logging.c File Reference	145
8.44.1	Detailed Description	146
8.44.2	Function Documentation	146
8.44.2.1	tests_log_single_test	146
8.44.3	Variable Documentation	146
8.44.3.1	show_failures	147
8.44.3.2	test_failures	147
8.44.3.3	test_successes	147
8.44.3.4	total_tests	147
8.45	src/vector.c File Reference	147
8.45.1	Detailed Description	148
8.45.2	Function Documentation	149
8.45.2.1	vector_insert_internal	149
8.45.3	Variable Documentation	149
8.45.3.1	GROWTH	149

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 command line parsing functionality	11
Public interface to string data structure	14
Private functionality for manipulating generic datatypes	24
Public general generic data structures functionality	28
Public interface to generic list data structure	33
Public interface to logging functionality	40
Public interface to generic queue data structure	41
Public interface to generic stack data structure	45
General purpose string manipulation functions	49
Public interface to unit testing functionality	53
Public interface to generic vector data structure.	59

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

dict	65
gds_kvpair	66
GDString	67
gdt_generic_datatype	
Generic datatype structure	68
list	71
list_node	72
list_string	
Structure to hold a list of strings	73
pair_string	
Structure to hold a string pair	74
queue	75
stack	77
vector	78

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

include/private/pggds_internal/ gds_common.h	
Common internal headers for data structures	81
include/private/pggds_internal/ gdt.h	
Interface to generic data element functionality	82
include/public/pggds/ dict.h	
Interface to generic dictionary data structure	84
include/public/pggds/ gds_opt.h	
Interface to command line option functions	87
include/public/pggds/ gds_public_types.h	
Common public types for generic data structures library	88
include/public/pggds/ gds_string.h	
Interface to string data structure	89
include/public/pggds/ gds_util.h	
Interface to general utility functions	92
include/public/pggds/ gds_util_error.h	
Interface to general utility error functions	93
include/public/pggds/ gds_util_logging.h	
Interface to logging functions	94
include/public/pggds/ gds_util_std_wrappers.h	
Interface to wrappers for standard functions	96
include/public/pggds/ gds_util_string.h	
Interface to general utility string functions	99
include/public/pggds/ kvpair.h	
Interface to generic key-value pair structure	99
include/public/pggds/ list.h	
Interface to generic list data structure	102
include/public/pggds/ queue.h	
Interface to generic queue data structure	104
include/public/pggds/ stack.h	
Interface to generic stack data structure	106
include/public/pggds/ string_util.h	
Interface to string utility functions	108
include/public/pggds/ test_logging.h	
Interface to unit test logging functionality	110
include/public/pggds/ unittest.h	
Public interface to unit test functionality	111
include/public/pggds/ vector.h	
Interface to generic vector data structure	112

src/ dict.c	Implementation of generic dictionary data structure	114
src/ gds_opt.c	Implementation of command line option functions	118
src/ gds_string.c	Implementation of string data structure	121
src/ gds_util_error.c	Implementation of general utility error functions	126
src/ gds_util_logging.c	Implementation of logging functions	127
src/ gds_util_std_wrappers.c	Implementation of wrappers for standard functions	128
src/ gdt.c	Implementation of generic data element functionality	131
src/ kvpair.c	Implementation of generic key-value pair structure	136
src/ list.c	Implementation of generic list data structure	138
src/ queue.c	Implementation of generic queue data structure	141
src/ stack.c	Implementation of generic stack data structure	142
src/ string_util.c	Implementation of string utility functions	144
src/ test_logging.c	Implementation of unit test logging functionality	145
src/ vector.c	Implementation of generic vector data structure	147

Chapter 6

Module Documentation

6.1 Public interface to command line parsing functionality

Functions

- bool [gds_parse_options](#) (const char *allowed, char **argv)
Parses a command line for options and non-options.
- void [gds_free_options](#) (void)
Frees memory associated with command line options.
- const char * [gds_option_prognam](#) (void)
Returns the program name.
- bool [gds_option_present](#) (const char *optname)
Checks if an option was provided on the command line.
- const char * [gds_option_argument_string](#) (const char *optname)
Retrieves a string argument for an option.
- bool [gds_option_argument_int](#) (const char *optname, int *value)
Retrieves an integer argument for an option.
- int [gds_option_nonopts_number](#) (void)
Returns the number of non-option arguments provided.
- const char * [gds_option_nonopt](#) (const size_t index)
Retrieves a non-option argument.

6.1.1 Detailed Description

This module contains functionality for parsing a command line and retrieving options, arguments to those options, and non-option command line arguments.

6.1.2 Function Documentation

6.1.2.1 void gds_free_options (void)

Frees memory associated with command line options.

6.1.2.2 bool gds_option_argument_int (const char * optname, int * value)

Retrieves an integer argument for an option.

Parameters

<i>optname</i>	A string containing the single-character option name.
<i>value</i>	A pointer to an <code>int</code> in which to store the value of the argument. If this is <code>NULL</code> , the function merely checks whether an argument representable by an integer is present.

Return values

<i>non-NULL</i>	The string argument
<i>NULL</i>	Failure, argument was not provided, argument could not be represented as an integer, or command line has not yet been parsed.

6.1.2.3 `const char* gds_option_argument_string (const char * optname)`

Retrieves a string argument for an option.

Parameters

<i>optname</i>	A string containing the single-character option name.
----------------	---

Return values

<i>non-NULL</i>	The string argument
<i>NULL</i>	Failure, argument was not provided, or command line has not yet been parsed.

6.1.2.4 `const char* gds_option_nonopt (const size_t index)`

Retrieves a non-option argument.

Parameters

<i>index</i>	The zero-based index of the non-option argument.
--------------	--

Returns

`non-NULL` The non-option argument at the specified index
`NULL` Index out-of-range, or command line has not yet been parsed

6.1.2.5 `int gds_option_nonopts_number (void)`

Returns the number of non-option arguments provided.

A non-option argument is any command line argument not of the form `-x`, where `x` is any alphanumeric character.

Returns

The number of non-option arguments provided. Zero is returned if the command line has not yet been parsed.

6.1.2.6 `bool gds_option_present (const char * optname)`

Checks if an option was provided on the command line.

Parameters

<i>optname</i>	A string containing the single-character option name.
----------------	---

Return values

<i>true</i>	Option was provided
<i>false</i>	Option was not provided, or command line has not yet been parsed.

6.1.2.7 `const char* gds_option_prognam (void)`

Returns the program name.

The string returned is equivalent to `argv[0]`.

Return values

<i>non-NULL</i>	The program name
<i>NULL</i>	Error, or command line has not yet been parsed, or program name was not present.

6.1.2.8 `bool gds_parse_options (const char * allowed, char ** argv)`

Parses a command line for options and non-options.

Parameters

<i>allowed</i>	A string containing the allowed options. Each option should be specified by a single alphabetic character. A ':' after an option signifies that it can take an argument.
<i>argv</i>	List of command line strings passed to <code>main()</code> .

Return values

<i>true</i>	Success
<i>false</i>	Failure, due to memory allocation failure, or badly-specified allowed options string, or unrecognized options

6.2 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](#) ([GDSSString](#) str)
Calculates a hash of a string.
- int [gds_str_compare](#) ([GDSSString](#) s1, [GDSSString](#) s2)
Compares two strings.
- int [gds_str_compare_cstr](#) ([GDSSString](#) s1, const char *s2)
Compares a string with a C-style string.
- int [gds_str_strchr](#) ([GDSSString](#) str, const char ch, const int start)
Returns index of first occurrence of a character.
- [GDSSString gds_str_substr_left](#) ([GDSSString](#) str, const size_t numchars)
Returns a left substring.
- [GDSSString gds_str_substr_right](#) ([GDSSString](#) str, const size_t numchars)
Returns a right substring.
- void [gds_str_split](#) ([GDSSString](#) src, [GDSSString](#) *left, [GDSSString](#) *right, const char sc)

- Splits a string.*
- void [gds_str_trim_leading](#) ([GDSSString](#) str)
Trims leading whitespace in-place.
- void [gds_str_trim_trailing](#) ([GDSSString](#) str)
Trims trailing whitespace in-place.
- void [gds_str_trim](#) ([GDSSString](#) str)
Trims leading and trailing whitespace in-place.
- char [gds_str_char_at_index](#) ([GDSSString](#) str, const size_t index)
Returns the character at a specified index.
- bool [gds_str_is_empty](#) ([GDSSString](#) str)
Checks if a string is empty.
- bool [gds_str_is_alnum](#) ([GDSSString](#) str)
Checks if a string contains only alphanumeric characters.
- void [gds_str_clear](#) ([GDSSString](#) str)
Clears (empties) a string.
- bool [gds_str_intval](#) ([GDSSString](#) str, const int base, int *value)
Gets the integer value of a string.
- bool [gds_str_doubleval](#) ([GDSSString](#) str, double *value)
Gets the double value of a string.
- [GDSSString](#) [gds_str_getline](#) (const size_t size, FILE *fp)
Gets a line from a file creates a new string.
- [GDSSString](#) [gds_str_getline_assign](#) ([GDSSString](#) str, const size_t size, FILE *fp)
Gets a line from a file and assigns it to a string.
- [GDSSString](#) [gds_str_decorate](#) ([GDSSString](#) str, [GDSSString](#) left_dec, [GDSSString](#) right_dec)
Brackets a string with decoration strings.

6.2.1 Detailed Description

A string is an ordered collection of characters.

6.2.2 Typedef Documentation

6.2.2.1 typedef struct [GDSSString](#)* [GDSSString](#)

Opaque data type for string.

6.2.3 Function Documentation

6.2.3.1 [GDSSString](#) [gds_str_assign](#) ([GDSSString](#) dst, [GDSSString](#) 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.

6.2.3.2 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.

6.2.3.3 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.

6.2.3.4 void gds_str_clear (GDSString *str*)

Clears (empties) a string.

Parameters

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

6.2.3.5 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*.

6.2.3.6 int gds_str_compare_cstr (GDSString *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*.

6.2.3.7 **GDSString gds_str_concat (GDSString *dst*, GDSString *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.

6.2.3.8 **GDSString gds_str_concat_cstr (GDSString *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.

6.2.3.9 **GDSString 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.

6.2.3.10 **GDSString 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.

6.2.3.11 `GDSString gds_str_create_sprintf (const char * format, ...)`

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

Parameters

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

Returns

The new string, or `NULL` on failure.

6.2.3.12 `const char* gds_str_cstr (GDSString 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.

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

Returns

The decorated string.

6.2.3.14 `void gds_str_destroy (GDSString str)`

Destroys a string and releases allocated resources.

Parameters

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

6.2.3.15 `bool gds_str_doubleval (GDSSString 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.

6.2.3.16 `GDSSString gds_str_dup (GDSSString src)`

Creates a new string from another string.

Parameters

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

Returns

The new string, or `NULL` on failure.

6.2.3.17 `GDSSString gds_str_getline (const size_t size, FILE * fp)`

Gets a line from a file creates a new string.

Any trailing newline character is stripped.

Parameters

<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`

6.2.3.18 `GDSSString gds_str_getline_assign (GDSSString 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`

6.2.3.19 unsigned long gds_str_hash (GDSSString *str*)

Calculates a hash of a string.

Uses Dan Bernstein's djb2 algorithm.

Parameters

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

Returns

The hash value

6.2.3.20 bool gds_str_intval (GDSSString *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.

6.2.3.21 bool gds_str_is_alnum (GDSSString *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.

6.2.3.22 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.

6.2.3.23 `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.

6.2.3.24 `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.

6.2.3.25 `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.

6.2.3.26 `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.

6.2.3.27 GDSString gds_str_substr_left (GDSString 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.

6.2.3.28 GDSString gds_str_substr_right (GDSString 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.

6.2.3.29 void gds_str_trim (GDSString str)

Trims leading and trailing whitespace in-place.

Parameters

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

6.2.3.30 void gds_str_trim_leading (GDSString str)

Trims leading whitespace in-place.

Parameters

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

6.2.3.31 void gds_str_trim_trailing (GDSString str)

Trims trailing whitespace in-place.

Parameters

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

6.2.3.32 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.

6.2.3.33 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.
------------	------------------------

6.3 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_STRING](#),
[DATATYPE_GDSSTRING](#), [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.

6.3.1 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.

6.3.2 Typedef Documentation

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

Type definition for comparison function pointer.

6.3.3 Enumeration Type Documentation

6.3.3.1 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_GDSSTRING [GDSString](#)
DATATYPE_POINTER void *

6.3.4 Function Documentation

6.3.4.1 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.

6.3.4.2 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.

6.3.4.3 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.
-------------	------------------------------------

6.3.4.4 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.

6.3.4.5 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.

6.3.4.6 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.

6.4 Public general generic data structures functionality

Macros

- `#define log_strerror(prog,...)`
Prints an error message with error number.
- `#define log_error(prog,...)`
Prints an error message.
- `#define quit_strerror(prog,...)`
Prints an error message with error number and exits.
- `#define quit_error(prog,...)`
Prints an error message and exits.
- `#define abort_error(prog,...)`
Prints an error message and aborts.
- `#define gds_assert(cond, prog,...)`
Tests an assertion and aborts on failure.
- `#define xmalloc(s) gds_xmalloc((s), __FILE__, __LINE__)`
Macro to call malloc() and abort on failure.
- `#define xcalloc(n, s) gds_xcalloc((n), (s), __FILE__, __LINE__)`
Macro to call calloc() and abort on failure.
- `#define xrealloc(p, s) gds_xrealloc((p), (s), __FILE__, __LINE__)`
Macro to call realloc() and abort on failure.
- `#define xstrdup(str) gds_xstrdup((str), __FILE__, __LINE__)`
Macro to call strdup() and abort on failure.
- `#define xfopen(path, mode) gds_xfopen((path), (mode), __FILE__, __LINE__)`
Macro to call strdup() and abort on failure.

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_logerror_line (const char *progrname, const char *filename, const int linenum, const bool log_errno, const enum gds_error_quit_type quit_type, const char *fmt,...)`
Logs an error message.
- `char * gds_strdup (const char *str)`
Dynamically duplicates a string.

6.4.1 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.

6.4.2 Macro Definition Documentation

6.4.2.1 #define abort_error(prog, ...)

Value:

```
gds_logerror_line((prog), \
    __FILE__, __LINE__, false, GDS_ERROR_ABORT, __VA_ARGS__)
```

Prints an error message and aborts.

Parameters

<i>prog</i>	The program name to include in the error message.
...	Other arguments, the first of which should be a format string suitable for passing to <code>vprintf()</code> , optionally followed by any additional arguments specified by the format string.

6.4.2.2 #define gds_assert(cond, prog, ...)

Value:

```
if ( !(cond) ) \
    gds_logerror_line((prog), __FILE__, __LINE__, \
        false, GDS_ERROR_ASSERT, __VA_ARGS__)
```

Tests an assertion and aborts on failure.

Parameters

<i>cond</i>	The assertion to test.
<i>prog</i>	The program name to include in the error message.
...	Other arguments, the first of which should be a format string suitable for passing to <code>vprintf()</code> , optionally followed by any additional arguments specified by the format string.

6.4.2.3 #define log_error(prog, ...)

Value:

```
gds_logerror_line((prog), \
    __FILE__, __LINE__, false, GDS_ERROR_NOQUIT, \
    __VA_ARGS__)
```

Prints an error message.

Parameters

<i>prog</i>	The program name to include in the error message.
...	Other arguments, the first of which should be a format string suitable for passing to <code>vprintf()</code> , optionally followed by any additional arguments specified by the format string.

6.4.2.4 #define log_strerror(prog, ...)

Value:

```
gds_logerror_line((prog), \
    __FILE__, __LINE__, true, GDS_ERROR_NOQUIT, __VA_ARGS__
)
```

Prints an error message with error number.

This macro 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. This function is intended to be called from the corresponding macro.

Parameters

<i>prog</i>	The program name to include in the error message.
...	Other arguments, the first of which should be a format string suitable for passing to <code>vprintf()</code> , optionally followed by any additional arguments specified by the format string.

6.4.2.5 `#define quit_error(prog, ...)`

Value:

```
gds_logerror_line((prog), \
    __FILE__, __LINE__, false, GDS_ERROR_EXIT, __VA_ARGS__)
```

Prints an error message and exits.

Parameters

<i>prog</i>	The program name to include in the error message.
...	Other arguments, the first of which should be a format string suitable for passing to <code>vprintf()</code> , optionally followed by any additional arguments specified by the format string.

6.4.2.6 `#define quit_strerror(prog, ...)`

Value:

```
gds_logerror_line((prog), \
    __FILE__, __LINE__, true, GDS_ERROR_EXIT, __VA_ARGS__)
```

Prints an error message with error number and exits.

This macro 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. This function is intended to be called from the corresponding macro.

Parameters

<i>prog</i>	The program name to include in the error message.
...	Other arguments, the first of which should be a format string suitable for passing to <code>vprintf()</code> , optionally followed by any additional arguments specified by the format string.

6.4.2.7 `#define xalloc(n, s) gds_xalloc((n), (s), __FILE__, __LINE__)`

Macro to call `calloc()` and abort on failure.

Parameters

<i>n</i>	The number of members to allocate.
<i>s</i>	The size in bytes of each member.

6.4.2.8 `#define xfdopen(path, mode) gds_xfdopen((path), (mode), __FILE__, __LINE__)`

Macro to call `strdup()` and abort on failure.

Parameters

<i>path</i>	The path of the file to open.
<i>mode</i>	The mode under which to open the file.

6.4.2.9 `#define xmalloc(s) gds_xmalloc((s), __FILE__, __LINE__)`

Macro to call `malloc()` and abort on failure.

Parameters

<i>s</i>	The number of bytes to allocate.
----------	----------------------------------

6.4.2.10 `#define xrealloc(p, s) gds_xrealloc((p), (s), __FILE__, __LINE__)`

Macro to call `realloc()` and abort on failure.

Parameters

<i>p</i>	A pointer to the memory to reallocate.
<i>s</i>	The number of bytes in the new allocation.

6.4.2.11 `#define xstrdup(str) gds_xstrdup((str), __FILE__, __LINE__)`

Macro to call `strdup()` and abort on failure.

Parameters

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

6.4.3 Enumeration Type Documentation

6.4.3.1 `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

6.4.4 Function Documentation

6.4.4.1 `void gds_logerror_line (const char * progrname, const char * filename, const int linenum, const bool log_erno, const enum gds_error_quit_type quit_type, const char * fmt, ...)`

Logs an error message.

This function is intended to be called via the accompanying macros.

Parameters

<i>prognose</i>	The program name to include in the message.
<i>filename</i>	The name of the source file.
<i>linenum</i>	The line number of the source file.
<i>log_errno</i>	Set to <code>true</code> to include the current value of <code>errno</code> and the string representation of that error in the message.
<i>quit_type</i>	Info on how to quit the function.
<i>fmt</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.

6.4.4.2 `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

6.5 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.

6.5.1 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).

6.5.2 Typedef Documentation

6.5.2.1 typedef struct list* List

Opaque list type definition.

6.5.2.2 typedef struct list_node* ListItr

Opaque list iterator type definition.

6.5.3 Function Documentation

6.5.3.1 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.

6.5.3.2 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.

6.5.3.3 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.

6.5.3.4 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.

6.5.3.5 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.

6.5.3.6 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.
-------------	------------------------

6.5.3.7 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.

6.5.3.8 `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

6.5.3.9 `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.

6.5.3.10 `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.

6.5.3.11 `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.

6.5.3.12 `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

6.5.3.13 `ListItr 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

6.5.3.14 `ListItr 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

6.5.3.15 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

6.5.3.16 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

6.5.3.17 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.

6.5.3.18 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.

6.5.3.19 `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.

6.5.3.20 `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.

6.5.3.21 `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.

6.6 Public interface to logging functionality

Functions

- FILE * [gds_errlog](#) (void)
Returns a pointer to the current log file.
- bool [gds_logging_on](#) (const char *logfilename, const bool append)
Starts logging functionality.
- bool [gds_logging_off](#) (void)
Stops logging functionality.

6.6.1 Detailed Description

This module contains functionality for logging to standard error or to a designated file, as well as macros for debug output.

6.6.2 Function Documentation

6.6.2.1 FILE* [gds_errlog](#) (void)

Returns a pointer to the current log file.

Returns

A pointer to the current log file.

6.6.2.2 bool [gds_logging_off](#) (void)

Stops logging functionality.

After calling this function, any calls to [gds_log_msg\(\)](#) will result in no action.

Return values

<i>true</i>	Success
<i>false</i>	Log file could not be closed, logging still stopped

6.6.2.3 bool [gds_logging_on](#) (const char * *logfilename*, const bool *append*)

Starts logging functionality.

Prior to calling this function, any calls to [gds_log_msg\(\)](#) will result in no action.

Parameters

<i>logfilename</i>	The name of the log file to open for writing, or NULL to log to the standard error stream.
<i>append</i>	Set to <i>true</i> to append to an existing log file, or <i>false</i> to overwrite it. This parameter is ignored if <i>logfilename</i> is NULL.

Return values

<i>true</i>	Success
<i>false</i>	Failure, log file could not be opened for writing

6.7 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.

6.7.1 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.

6.7.2 Typedef Documentation

6.7.2.1 typedef struct queue* Queue

Opaque queue type definition.

6.7.3 Function Documentation

6.7.3.1 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.

6.7.3.2 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.

6.7.3.3 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.
--------------	-------------------------

6.7.3.4 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.

6.7.3.5 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

6.7.3.6 `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

6.7.3.7 `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.

6.7.3.8 `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.

6.7.3.9 `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

6.7.3.10 `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.

6.8 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.

6.8.1 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.

6.8.2 Typedef Documentation

6.8.2.1 typedef struct `stack`* `Stack`

Opaque stack type definition.

6.8.3 Function Documentation

6.8.3.1 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.

6.8.3.2 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.

6.8.3.3 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.
--------------	-------------------------

6.8.3.4 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.

6.8.3.5 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

6.8.3.6 `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

6.8.3.7 `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.

6.8.3.8 `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.

6.8.3.9 `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.

6.8.3.10 `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.

6.9 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.

6.9.1 Detailed Description

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

6.9.2 Function Documentation

6.9.2.1 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

6.9.2.2 `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

6.9.2.3 `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.

6.9.2.4 `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.

6.9.2.5 `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.

6.9.2.6 `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.

6.9.2.7 `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

6.9.2.8 `void list_string_destroy (struct list_string * list)`

Destroys a string list.

Parameters

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

6.9.2.9 `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

6.9.2.10 `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

6.9.2.11 `void pair_string_destroy (struct pair_string * pair)`

Destroys a string pair.

Parameters

<i>pair</i>	The pair to destroy.
-------------	----------------------

6.9.2.12 `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

6.10 Public interface to unit testing functionality

Macros

- `#define TEST_SUITE(name)`
Macro for defining a test suite.
- `#define TEST_CASE(name)`
Macro for defining a test case.
- `#define RUN_CASE(name) name(name##_testcasename)`
Macro to run a test case.
- `#define TEST_ASSERT_TRUE(cond)`
Macro to test if a given condition is true.
- `#define TEST_ASSERT_FALSE(cond)`
Macro to test if a given condition is false.
- `#define TEST_ASSERT_EQUAL(a, b)`
Macro to test if two values are equal.
- `#define TEST_ASSERT_NOTEQUAL(a, b)`
Macro to test if two values are not equal.
- `#define TEST_ASSERT_ALMOST_EQUAL(a, b, e)`
Macro to test two real numbers for fuzzy equality.
- `#define TEST_ASSERT_STR_EQUAL(s1, s2)`
Macro to test if two strings are equal.
- `#define TEST_ASSERT_STR_NOTEQUAL(s1, s2)`
Macro to test if two strings are not equal.

Functions

- `void tests_assert_true (const bool success, const char *suiteName, const char *casename, const char *failmessage, const char *filename, const int linenum)`
Logs the result of a true/false unit test.
- `bool tests_assert_almost_equal (const long double a, const long double b, const long double e)`
Tests two real numbers for fuzzy equality.
- `void tests_initialize (void)`
Initializes the test runner.
- `void tests_report (void)`
Reports on the test results.
- `int tests_get_total_tests (void)`
Returns the total number of tests run.
- `int tests_get_successes (void)`
Returns the total number of successful tests.
- `int tests_get_failures (void)`
Returns the total number of failed tests.

6.10.1 Detailed Description

Unit testing macros and functions.

6.10.2 Macro Definition Documentation

6.10.2.1 `#define RUN_CASE(name) name(name##_testcasename)`

Macro to run a test case.

Parameters

<i>name</i>	The name of the test case, as previously defined by a call to TEST_CASE() .
-------------	---

6.10.2.2 `#define TEST_ASSERT_ALMOST_EQUAL(a, b, e)`

Value:

```
tests_assert_true( \
    tests_assert_almost_equal(a, b, e), \
    izzywig_testsuitename, \
    izzywig_testcasename, \
    (#a " is not fuzzily equal to " #b), \
    __FILE__, \
    __LINE__)
```

Macro to test two real numbers for fuzzy equality.

Parameters

<i>a</i>	The first number.
<i>b</i>	The second number.
<i>e</i>	The equality threshold. The first parameter will be multiplied by this quantity (unless (a) the first parameter is zero, in which case the second parameter will be multiplied by it; or (b) both parameters are zero, in which case the function will return true) and the function will be true if the absolute difference between the first two parameters is smaller than or equal to this value.

6.10.2.3 `#define TEST_ASSERT_EQUAL(a, b)`

Value:

```
tests_assert_true((a==(b)), \
    izzywig_testsuitename, \
    izzywig_testcasename, \
    (#a " is not equal to " #b), \
    __FILE__, \
    __LINE__)
```

Macro to test if two values are equal.

Parameters

<i>a</i>	The first value.
<i>b</i>	The second value.

6.10.2.4 `#define TEST_ASSERT_FALSE(cond)`

Value:

```
tests_assert_true(!cond), \
    izzywig_testsuitename, \
    izzywig_testcasename, \
    (#cond " is not false"), \
```

```
__FILE__, \
__LINE__)
```

Macro to test if a given condition is false.

Parameters

<i>cond</i>	The condition to test.
-------------	------------------------

6.10.2.5 #define TEST_ASSERT_NOTEQUAL(a, b)

Value:

```
tests_assert_true((a) != (b)), \
    izzywig_testsuitename, \
    izzywig_testcasename, \
    (#a " is equal to " #b), \
    __FILE__, \
    __LINE__)
```

Macro to test if two values are not equal.

Parameters

<i>a</i>	The first value.
<i>b</i>	The second value.

6.10.2.6 #define TEST_ASSERT_STR_EQUAL(s1, s2)

Value:

```
tests_assert_true(!strcmp((s1), (s2)), \
    izzywig_testsuitename, \
    izzywig_testcasename, \
    (#s1 " is not equal to " #s2), \
    __FILE__, \
    __LINE__)
```

Macro to test if two strings are equal.

Parameters

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

6.10.2.7 #define TEST_ASSERT_STR_NOTEQUAL(s1, s2)

Value:

```
tests_assert_true(strcmp((s1), (s2)), \
    izzywig_testsuitename, \
    izzywig_testcasename, \
    (#s1 " is equal to " #s2), \
    __FILE__, \
    __LINE__)
```

Macro to test if two strings are not equal.

Parameters

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

6.10.2.8 `#define TEST_ASSERT_TRUE(cond)`

Value:

```
tests_assert_true((cond), \
    izzywig_testsuitename, \
    izzywig_testcasename, \
    (#cond " is not true"), \
    __FILE__, \
    __LINE__)
```

Macro to test if a given condition is true.

Parameters

<i>cond</i>	The condition to test.
-------------	------------------------

6.10.2.9 `#define TEST_CASE(name)`

Value:

```
static const char * const \
    name##_testcasename = (#name); \
static void name(const char * const izzywig_testcasename)
```

Macro for defining a test case.

Parameters

<i>name</i>	The name of the test case.
-------------	----------------------------

6.10.2.10 `#define TEST_SUITE(name)`

Value:

```
static const char * const \
    izzywig_testsuitename = (#name)
```

Macro for defining a test suite.

This macro should be called prior to defining any test cases.

Parameters

<i>name</i>	The name of the test suite.
-------------	-----------------------------

6.10.3 Function Documentation

6.10.3.1 `bool tests_assert_almost_equal (const long double a, const long double b, const long double e)`

Tests two real numbers for fuzzy equality.

Parameters

<i>a</i>	The first number.
<i>b</i>	The second number.
<i>e</i>	The equality threshold. The first parameter will be multiplied by this quantity (unless (a) the first parameter is zero, in which case the second parameter will be multiplied by it; or (b) both parameters are zero, in which case the function will return true) and the function will be true if the absolute difference between the first two parameters is smaller than or equal to this value.

Return values

<i>true</i>	The numbers are equal to the specified precision
<i>false</i>	The numbers are not equal to the specified precision

6.10.3.2 `void tests_assert_true (const bool success, const char * suitename, const char * casename, const char * failmessage, const char * filename, const int linenum)`

Logs the result of a true/false unit test.

A message is output to standard error on test failure, showing the suite and case name, the source file and line of the test, and a message. This function is designed to be called via one of the TEST_ macros, and in most cases should not be called directly.

Parameters

<i>success</i>	The test condition.
<i>suitename</i>	The name of the test suite.
<i>casename</i>	The name of the test case.
<i>failmessage</i>	The message to print on test failure.
<i>filename</i>	The name of the file containing the test.
<i>linenum</i>	The source file line number containing the test.

6.10.3.3 `int tests_get_failures (void)`

Returns the total number of failed tests.

Returns

The total number of failed tests.

6.10.3.4 `int tests_get_successes (void)`

Returns the total number of successful tests.

Returns

The total number of successful tests.

6.10.3.5 `int tests_get_total_tests (void)`

Returns the total number of tests run.

Returns

The total number of tests run.

6.10.3.6 void tests_initialize (void)

Initializes the test runner.

6.10.3.7 void tests_report (void)

Reports on the test results.

6.11 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.

6.11.1 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).

6.11.2 Typedef Documentation

6.11.2.1 typedef struct vector* Vector

Opaque vector type definition.

6.11.3 Function Documentation

6.11.3.1 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.

6.11.3.2 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.

6.11.3.3 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.

6.11.3.4 `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.

6.11.3.5 `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.

6.11.3.6 `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.

6.11.3.7 `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.
---------------	--------------------------

6.11.3.8 `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.

6.11.3.9 `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

6.11.3.10 `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.

6.11.3.11 `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.

6.11.3.12 `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

6.11.3.13 `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.

6.11.3.14 `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.

6.11.3.15 void vector_reverse_sort (Vector vector)

Sorts a vector in-place, in descending order.

Parameters

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

6.11.3.16 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.
<i>...</i>	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.

6.11.3.17 void vector_sort (Vector vector)

Sorts a vector in-place, in ascending order.

Parameters

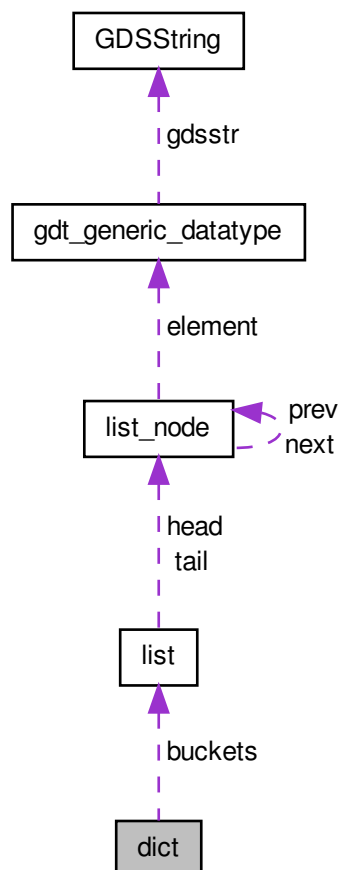
<i>vector</i>	A pointer to the vector.
---------------	--------------------------

Chapter 7

Data Structure Documentation

7.1 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

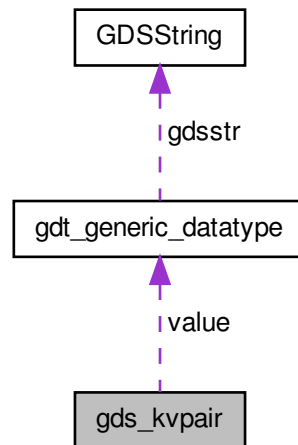
The documentation for this struct was generated from the following file:

- `src/dict.c`

7.2 `gds_kvpair` Struct Reference

```
#include <kvpair.h>
```

Collaboration diagram for gds_kvpair:



Data Fields

- char * [key](#)
- struct [gdt_generic_datatype](#) [value](#)

7.2.1 Detailed Description

Key-Value pair structure

7.2.2 Field Documentation

7.2.2.1 char* gds_kvpair::key

String key

7.2.2.2 struct [gdt_generic_datatype](#) gds_kvpair::value

Generic datatype value

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

- [include/public/pggds/kvpair.h](#)

7.3 GDSSString Struct Reference

Data Fields

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

- `size_t capacity`

7.3.1 Detailed Description

Structure to contain string

7.3.2 Field Documentation

7.3.2.1 `size_t GDSSString::capacity`

The size of the `data` buffer

7.3.2.2 `char* GDSSString::data`

The data in C-style string format

7.3.2.3 `size_t GDSSString::length`

The length of the string

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

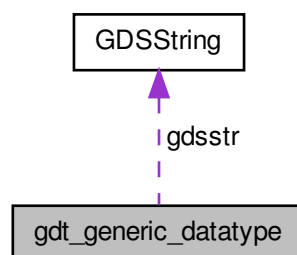
- `src/gds_string.c`

7.4 `gdt_generic_datatype` Struct Reference

Generic datatype structure.

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

Collaboration diagram for `gdt_generic_datatype`:



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](#)
 - [GDSSString gdsstr](#)
 - void * [p](#)

[} data](#)

7.4.1 Detailed Description

Generic datatype structure.

7.4.2 Field Documentation

7.4.2.1 char gdt_generic_datatype::c

char

7.4.2.2 gds_cfunc gdt_generic_datatype::compfunc

Comparison function pointer

7.4.2.3 double gdt_generic_datatype::d

double

7.4.2.4 union { ... } gdt_generic_datatype::data

Data union

7.4.2.5 GDSSString gdt_generic_datatype::gdsstr

[GDSSString](#)

7.4.2.6 int gdt_generic_datatype::i

int

7.4.2.7 long gdt_generic_datatype::l

long

7.4.2.8 `long long int gdt_generic_datatype::ll`

long long

7.4.2.9 `void* gdt_generic_datatype::p`

void *

7.4.2.10 `char* gdt_generic_datatype::pc`

char *, string

7.4.2.11 `signed char gdt_generic_datatype::sc`

signed char

7.4.2.12 `size_t gdt_generic_datatype::st`

size_t

7.4.2.13 `enum gds_datatype gdt_generic_datatype::type`

Data type

7.4.2.14 `unsigned char gdt_generic_datatype::uc`

unsigned char

7.4.2.15 `unsigned int gdt_generic_datatype::ui`

unsigned int

7.4.2.16 `unsigned long gdt_generic_datatype::ul`

unsigned long

7.4.2.17 `unsigned long long int gdt_generic_datatype::ull`

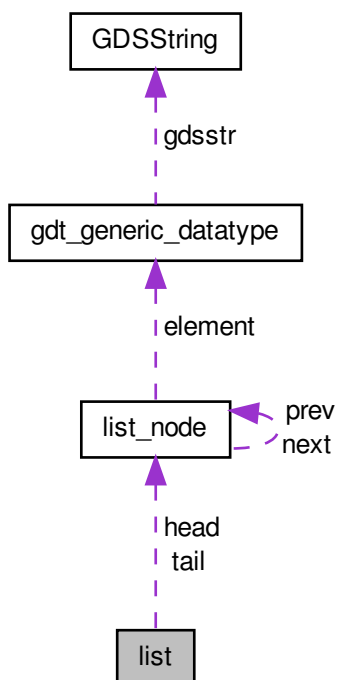
unsigned long long

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

- `include/private/pggds_internal/gdt.h`

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`

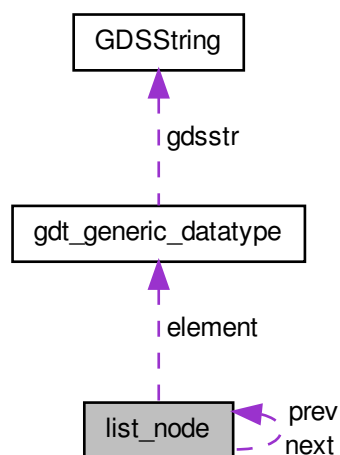
List datatype

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/pggds/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`

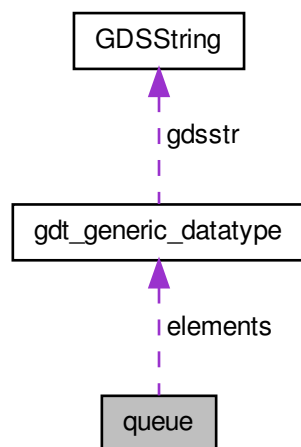
Second string of pair

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

- `include/public/pggds/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

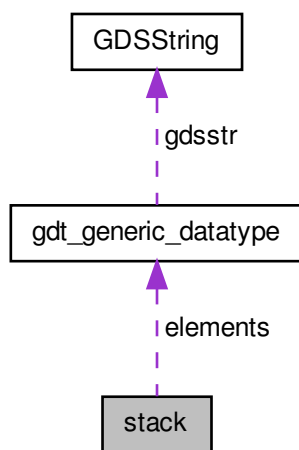
Queue datatype

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`

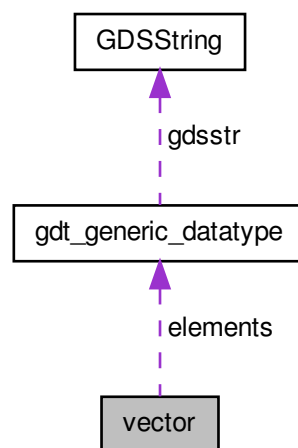
Stack datatype

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/cmdline.dox File Reference

8.2 docs/gds.dox File Reference

8.3 docs/gds_string.dox File Reference

8.4 docs/gdt.dox File Reference

8.5 docs/general.dox File Reference

8.6 docs/list.dox File Reference

8.7 docs/logging.dox File Reference

8.8 docs/queue.dox File Reference

8.9 docs/stack.dox File Reference

8.10 docs/string_util.dox File Reference

8.11 docs/unittest.dox File Reference

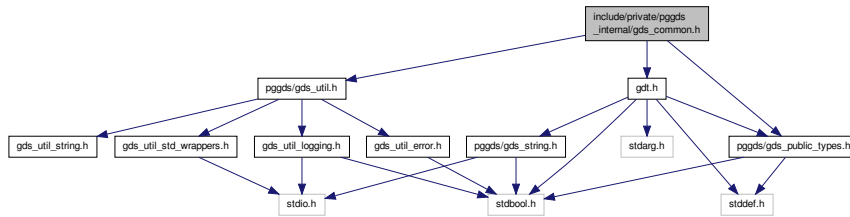
8.12 docs/vector.dox File Reference

8.13 include/private/pggds_internal/gds_common.h File Reference

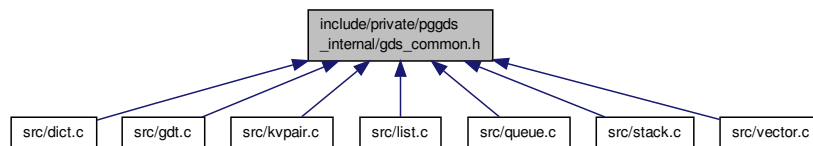
Common internal headers for data structures.

```
#include <pggds/gds_public_types.h>
#include <pggds/gds_util.h>
#include "gdt.h"
```

Include dependency graph for gds_common.h:



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



8.13.1 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/>

8.14 include/private/ppgds_internal/gdt.h File Reference

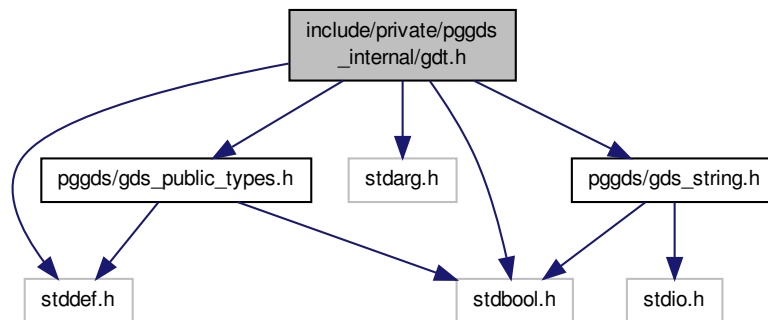
Interface to generic data element functionality.

```

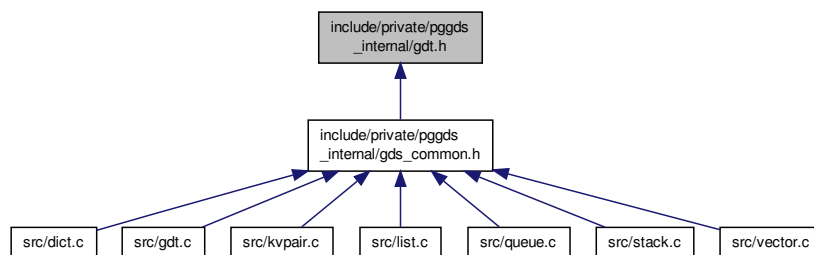
#include <stdbool.h>
#include <stddef.h>
#include <stdarg.h>
#include <pggds/gds_public_types.h>
#include <pggds/gds_string.h>

```

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.

8.14.1 Detailed Description

Interface to generic data element functionality.

Author

Paul Griffiths

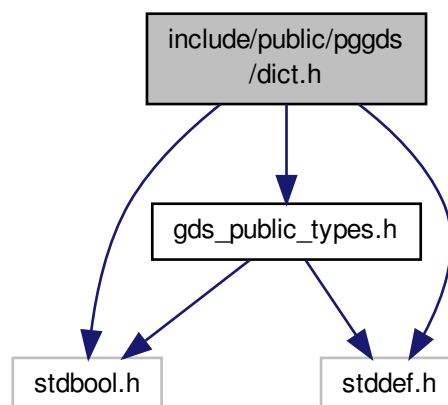
Copyright

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

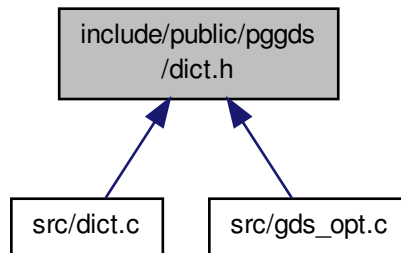
8.15 include/public/pggds/dict.h File Reference

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.

8.15.1 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/>

8.15.2 Typedef Documentation

8.15.2.1 typedef struct `dict`* `Dict`

Opaque dictionary type definition.

8.15.3 Function Documentation

8.15.3.1 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.

8.15.3.2 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.
-------------	------------------------------

8.15.3.3 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

8.15.3.4 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

8.15.3.5 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

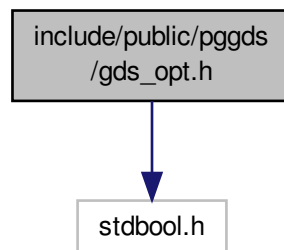
<i>true</i>	Success
<i>false</i>	Failure, key was not found

8.16 include/public/pggds/gds_opt.h File Reference

Interface to command line option functions.

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

Include dependency graph for gds_opt.h:



Functions

- bool [gds_parse_options](#) (const char *allowed, char **argv)
Parses a command line for options and non-options.
- void [gds_free_options](#) (void)
Frees memory associated with command line options.

- const char * [gds_option_progname](#) (void)
Returns the program name.
- bool [gds_option_present](#) (const char *optname)
Checks if an option was provided on the command line.
- const char * [gds_option_argument_string](#) (const char *optname)
Retrieves a string argument for an option.
- bool [gds_option_argument_int](#) (const char *optname, int *value)
Retrieves an integer argument for an option.
- int [gds_option_nonopts_number](#) (void)
Returns the number of non-option arguments provided.
- const char * [gds_option_nonopt](#) (const size_t index)
Retrieves a non-option argument.

8.16.1 Detailed Description

Interface to command line option functions.

Author

Paul Griffiths

Copyright

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

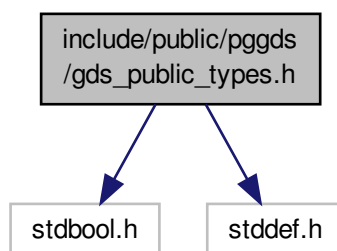
8.17 include/public/pggds/gds_public_types.h File Reference

Common public types for generic data structures library.

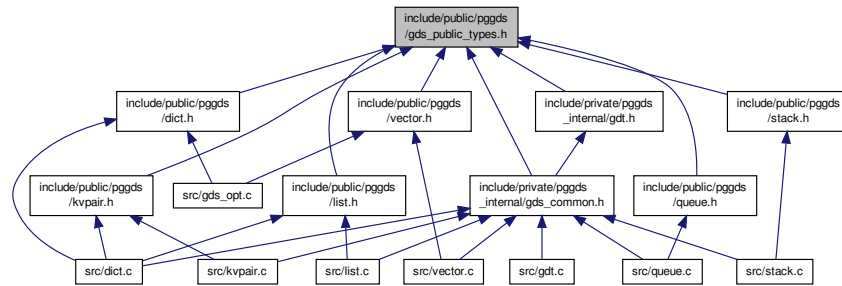
```
#include <stdbool.h>
```

```
#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_GDSSTRING](#), [DATATYPE_POINTER](#) }

Enumeration type for data element type.

8.17.1 Detailed Description

Common public types for generic data structures library.

Author

Paul Griffiths

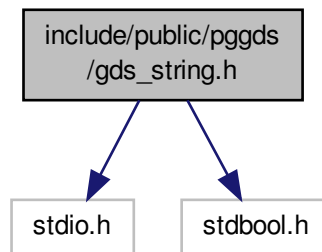
Copyright

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

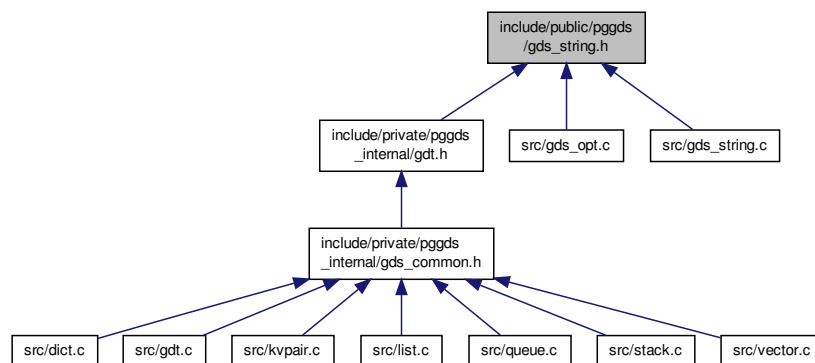
8.18 include/public/pggds/gds_string.h File Reference

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 [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](#) (GDSSString str)
Calculates a hash of a string.
- int [gds_str_compare](#) (GDSSString s1, GDSSString s2)
Compares two strings.
- int [gds_str_compare_cstr](#) (GDSSString s1, const char *s2)
Compares a string with a C-style string.
- int [gds_str_strchr](#) (GDSSString str, const char ch, const int start)
Returns index of first occurrence of a character.
- [GDSSString gds_str_substr_left](#) (GDSSString str, const size_t numchars)
Returns a left substring.
- [GDSSString gds_str_substr_right](#) (GDSSString str, const size_t numchars)
Returns a right substring.
- void [gds_str_split](#) (GDSSString src, GDSSString *left, GDSSString *right, const char sc)
Splits a string.
- void [gds_str_trim_leading](#) (GDSSString str)
Trims leading whitespace in-place.
- void [gds_str_trim_trailing](#) (GDSSString str)
Trims trailing whitespace in-place.
- void [gds_str_trim](#) (GDSSString str)
Trims leading and trailing whitespace in-place.
- char [gds_str_char_at_index](#) (GDSSString str, const size_t index)
Returns the character at a specified index.
- bool [gds_str_is_empty](#) (GDSSString str)
Checks if a string is empty.
- bool [gds_str_is_alnum](#) (GDSSString str)
Checks if a string contains only alphanumeric characters.
- void [gds_str_clear](#) (GDSSString str)
Clears (empties) a string.
- bool [gds_str_intval](#) (GDSSString str, const int base, int *value)
Gets the integer value of a string.
- bool [gds_str_doubleval](#) (GDSSString str, double *value)

Gets the double value of a string.

- `GDSString gds_str_getline` (const size_t size, FILE *fp)

Gets a line from a file creates a new string.

- `GDSString gds_str_getline_assign` (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.

8.18.1 Detailed Description

Interface to string data structure.

Author

Paul Griffiths

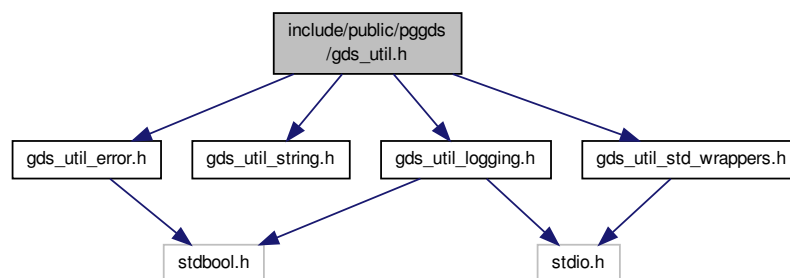
Copyright

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

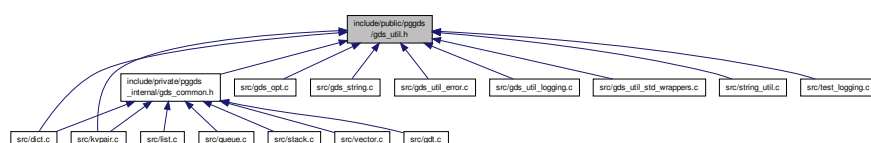
8.19 include/public/pggds/gds_util.h File Reference

Interface to general utility functions.

```
#include "gds_util_error.h"
#include "gds_util_string.h"
#include "gds_util_std_wrappers.h"
#include "gds_util_logging.h"
Include dependency graph for gds_util.h:
```



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



8.19.1 Detailed Description

Interface to general utility functions.

Author

Paul Griffiths

Copyright

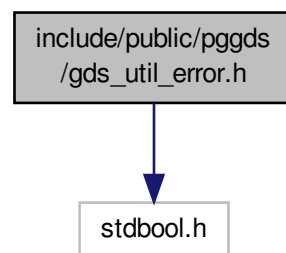
Copyright 2014 Paul Griffiths. Distributed under the terms of the GNU General Public License. <http://www.gnu.org/licenses/>

8.20 include/public/pggds/gds_util_error.h File Reference

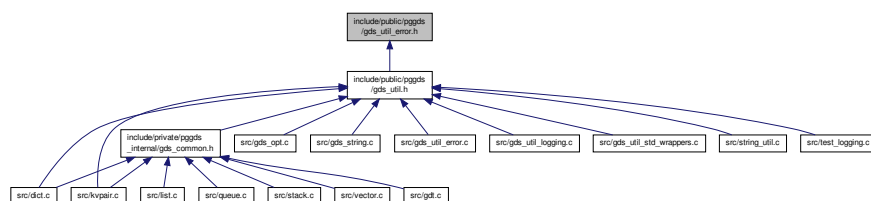
Interface to general utility error functions.

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

Include dependency graph for gds_util_error.h:



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



Macros

- `#define log_strerror(prog,...)`
Prints an error message with error number.
- `#define log_error(prog,...)`
Prints an error message.
- `#define quit_strerror(prog,...)`

Prints an error message with error number and exits.

- #define `quit_error`(prog,...)

Prints an error message and exits.

- #define `abort_error`(prog,...)

Prints an error message and aborts.

- #define `gds_assert`(cond, prog,...)

Tests an assertion and aborts on failure.

Enumerations

- enum `gds_error_quit_type` { `GDS_ERROR_NOQUIT`, `GDS_ERROR_EXIT`, `GDS_ERROR_ABORT`, `GDS_ERROR_ASSERT` }

Functions

- void `gds_logerror_line` (const char *`progrname`, const char *`filename`, const int `linenum`, const bool `log_errno`, const enum `gds_error_quit_type` `quit_type`, const char *`fmt`,...)

Logs an error message.

8.20.1 Detailed Description

Interface to general utility error functions.

Author

Paul Griffiths

Copyright

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

8.20.2 Enumeration Type Documentation

8.20.2.1 enum `gds_error_quit_type`

Enumerator:

`GDS_ERROR_NOQUIT`

`GDS_ERROR_EXIT`

`GDS_ERROR_ABORT`

`GDS_ERROR_ASSERT`

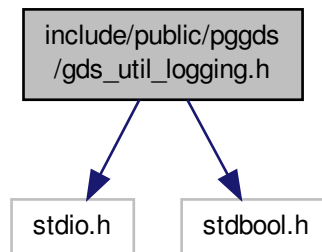
8.21 `include/public/pggds/gds_util_logging.h` File Reference

Interface to logging functions.

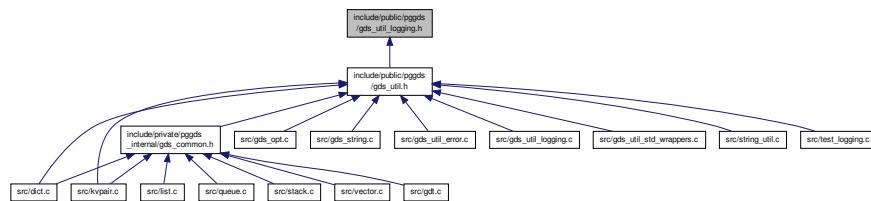
```
#include <stdio.h>
```

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

Include dependency graph for gds_util_logging.h:



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



Macros

- `#define DPRINTF(...)`
Debug printf macro.

Functions

- FILE * `gds_errlog` (void)
Returns a pointer to the current log file.
- bool `gds_logging_on` (const char *logfilename, const bool append)
Starts logging functionality.
- bool `gds_logging_off` (void)
Stops logging functionality.
- void `gds_log_msg` (const char *fmt,...)

8.21.1 Detailed Description

Interface to logging functions.

Author

Paul Griffiths

Copyright

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

8.21.2 Macro Definition Documentation

8.21.2.1 #define DPRINTF(...)

Debug printf macro.

Parameters

...	Arguments suitable for passing to printf()
-----	--

8.21.3 Function Documentation

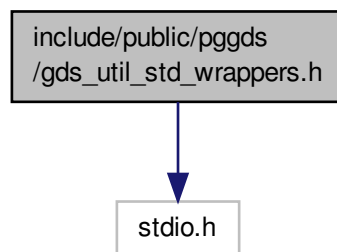
8.21.3.1 void gds_log_msg (const char * *fmt*, ...)

8.22 include/public/pggds/gds_util_std_wrappers.h File Reference

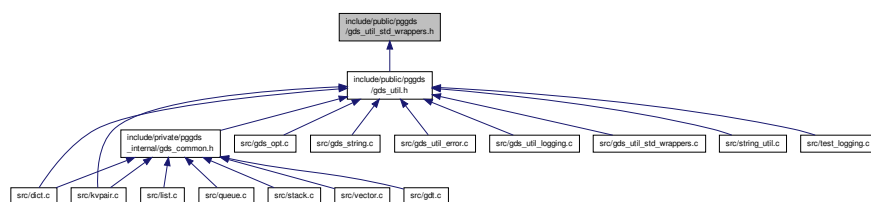
Interface to wrappers for standard functions.

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

Include dependency graph for gds_util_std_wrappers.h:



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



Macros

- #define `xmalloc(s)` `gds_xmalloc((s), __FILE__, __LINE__)`
Macro to call `malloc()` and abort on failure.
- #define `xcalloc(n, s)` `gds_xcalloc((n), (s), __FILE__, __LINE__)`
Macro to call `calloc()` and abort on failure.
- #define `xrealloc(p, s)` `gds_xrealloc((p), (s), __FILE__, __LINE__)`
Macro to call `realloc()` and abort on failure.
- #define `xstrdup(str)` `gds_xstrdup((str), __FILE__, __LINE__)`
Macro to call `strdup()` and abort on failure.
- #define `xfopen(path, mode)` `gds_xfopen((path), (mode), __FILE__, __LINE__)`
Macro to call `strdup()` and abort on failure.

Functions

- void * `gds_xmalloc` (const size_t size, const char *filename, const int linenum)
Wraps `malloc()` and aborts on failure.
- void * `gds_xcalloc` (const size_t nmemb, const size_t size, const char *filename, const int linenum)
Wraps `calloc()` and aborts on failure.
- void * `gds_xrealloc` (void *ptr, const size_t size, const char *filename, const int linenum)
Wraps `realloc()` and aborts on failure.
- char * `gds_xstrdup` (const char *str, const char *filename, const int linenum)
Wraps `strdup()` and aborts on failure.
- FILE * `gds_xfopen` (const char *path, const char *mode, const char *filename, const int linenum)
Wraps `fopen()` and exits on failure.

8.22.1 Detailed Description

Interface to wrappers for standard functions.

Author

Paul Griffiths

Copyright

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

8.22.2 Function Documentation

8.22.2.1 void* gds_xcalloc (const size_t nmemb, const size_t size, const char * filename, const int linenum)

Wraps `calloc()` and aborts on failure.

This is designed to be called from the corresponding macro.

Parameters

<i>nmemb</i>	The number of members to allocate.
<i>size</i>	The size in bytes of each member.
<i>filename</i>	The name of the calling file.
<i>linenum</i>	The line number in the calling file.

Returns

A pointer to the allocated memory.

8.22.2.2 FILE* gds_xfopen (const char * *path*, const char * *mode*, const char * *filename*, const int *linenum*)

Wraps fopen() and exits on failure.

This is designed to be called from the corresponding macro.

Parameters

<i>path</i>	The path of the file to open.
<i>mode</i>	The mode under which to open the file.
<i>filename</i>	The name of the calling file.
<i>linenum</i>	The line number in the calling file.

Returns

A pointer to the allocated memory.

8.22.2.3 void* gds_xmalloc (const size_t *size*, const char * *filename*, const int *linenum*)

Wraps malloc() and aborts on failure.

This is designed to be called from the corresponding macro.

Parameters

<i>size</i>	The number of bytes to allocate.
<i>filename</i>	The name of the calling file.
<i>linenum</i>	The line number in the calling file.

Returns

A pointer to the allocated memory.

8.22.2.4 void* gds_xrealloc (void * *ptr*, const size_t *size*, const char * *filename*, const int *linenum*)

Wraps realloc() and aborts on failure.

This is designed to be called from the corresponding macro.

Parameters

<i>ptr</i>	A pointer to the memory to reallocate.
<i>size</i>	The number of bytes for the new allocation.
<i>filename</i>	The name of the calling file.
<i>linenum</i>	The line number in the calling file.

Returns

A pointer to the reallocated memory.

8.22.2.5 char* gds_xstrdup (const char * str, const char * filename, const int linenum)

Wraps strdup() and aborts on failure.

This is designed to be called from the corresponding macro.

Parameters

<i>str</i>	The string to duplicate.
<i>filename</i>	The name of the calling file.
<i>linenum</i>	The line number in the calling file.

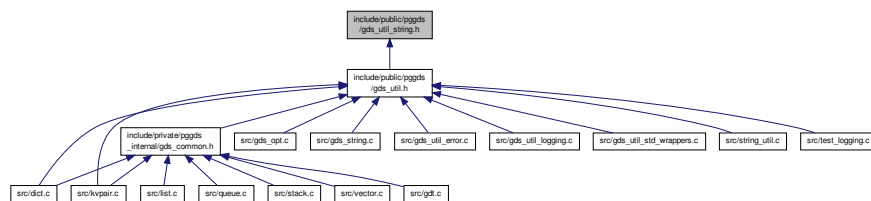
Returns

A pointer to the allocated memory.

8.23 include/public/pggds/gds_util_string.h File Reference

Interface to general utility string functions.

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



Functions

- char * [gds_strdup](#) (const char *str)
Dynamically duplicates a string.

8.23.1 Detailed Description

Interface to general utility string functions.

Author

Paul Griffiths

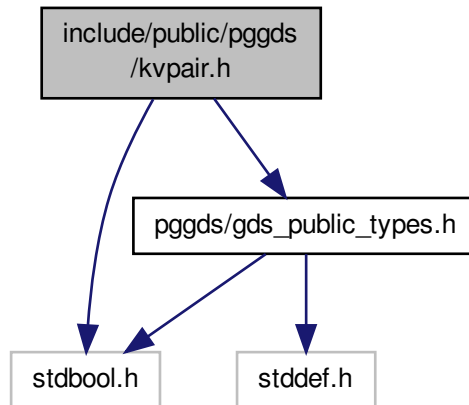
Copyright

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

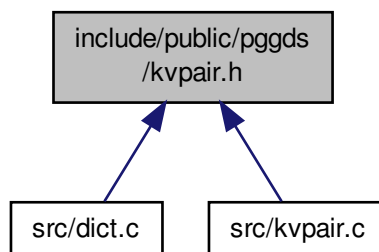
8.24 include/public/pggds/kvpair.h File Reference

Interface to generic key-value pair structure.

```
#include <stdbool.h>
#include <pggds/gds_public_types.h>
Include dependency graph for kvpair.h:
```



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



Data Structures

- struct [gds_kvpair](#)

Typedefs

- typedef struct [gds_kvpair](#) * [KVPair](#)

Functions

- [KVPair gds_kvpair_create](#) (const char *key, const enum [gds_datatype](#) type, va_list ap)

Creates a new key-value pair.

- void [gds_kvpair_destroy](#) ([KVPair](#) pair, const bool free_value)

Destroys a key-value pair.

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

Compares two key-value pairs by key.

8.24.1 Detailed Description

Interface to generic key-value pair structure.

Author

Paul Griffiths

Copyright

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

8.24.2 Typedef Documentation

8.24.2.1 typedef struct gds_kvpair * KVPair

Key-Value pair structure

8.24.3 Function Documentation

8.24.3.1 int gds_kvpair_compare (const void * p1, const void * p2)

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

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

8.24.3.2 KVPair gds_kvpair_create (const char * key, const enum gds_datatype type, va_list ap)

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

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

8.24.3.3 void gds_kvpair_destroy (KVPair pair, const bool free_value)

Destroys a key-value pair.

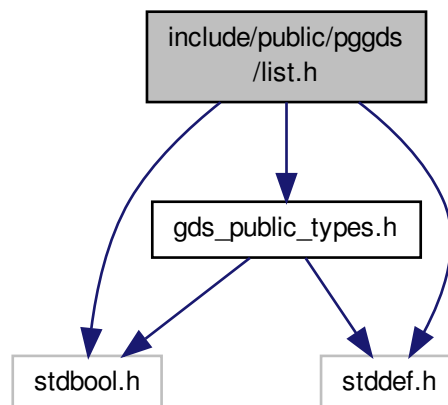
Parameters

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

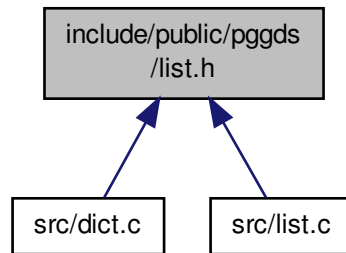
8.25 include/public/pggds/list.h File Reference

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](#) * [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.

8.25.1 Detailed Description

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

Author

Paul Griffiths

Copyright

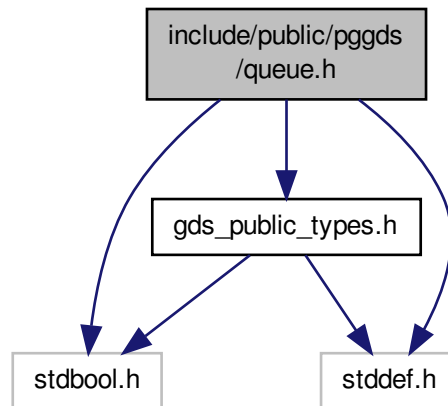
Copyright 2014 Paul Griffiths. Distributed under the terms of the GNU General Public License. <http://www.gnu.org/licenses/>

8.26 include/public/pggds/queue.h File Reference

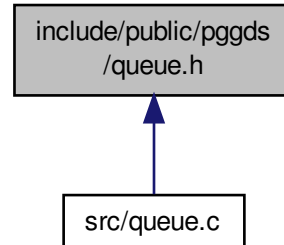
Interface to generic queue data structure.

```
#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.

8.26.1 Detailed Description

Interface to generic queue data structure.

Author

Paul Griffiths

Copyright

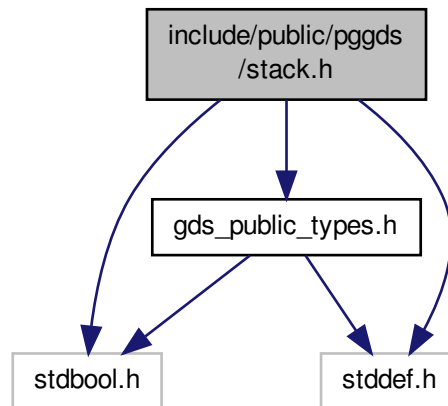
Copyright 2014 Paul Griffiths. Distributed under the terms of the GNU General Public License. <http://www.gnu.org/licenses/>

8.27 include/public/pggds/stack.h File Reference

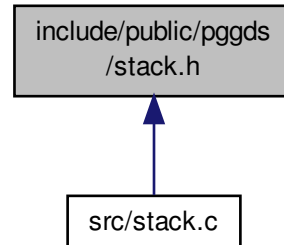
Interface to generic stack data structure.

```
#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.

8.27.1 Detailed Description

Interface to generic stack data structure.

Author

Paul Griffiths

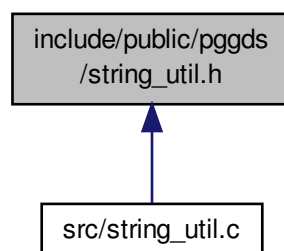
Copyright

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

8.28 include/public/pggds/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.

8.28.1 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/>

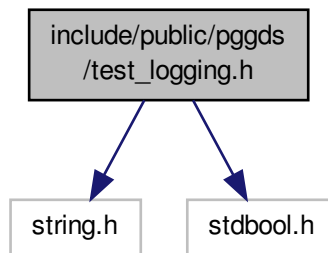
8.29 include/public/pggds/test_logging.h File Reference

Interface to unit test logging functionality.

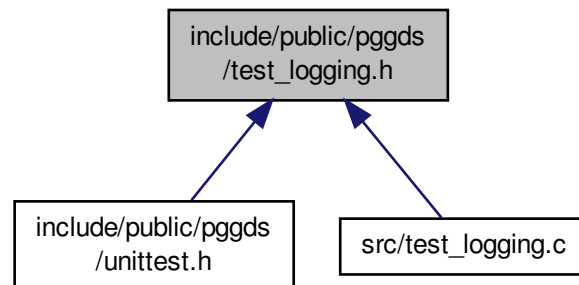
```
#include <string.h>
```

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

Include dependency graph for test_logging.h:



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



Macros

- `#define TEST_SUITE(name)`
Macro for defining a test suite.
- `#define TEST_CASE(name)`
Macro for defining a test case.
- `#define RUN_CASE(name) name(name##_testcasename)`
Macro to run a test case.
- `#define TEST_ASSERT_TRUE(cond)`
Macro to test if a given condition is true.
- `#define TEST_ASSERT_FALSE(cond)`
Macro to test if a given condition is false.

- #define `TEST_ASSERT_EQUAL(a, b)`
Macro to test if two values are equal.
- #define `TEST_ASSERT_NOTEQUAL(a, b)`
Macro to test if two values are not equal.
- #define `TEST_ASSERT_ALMOST_EQUAL(a, b, e)`
Macro to test two real numbers for fuzzy equality.
- #define `TEST_ASSERT_STR_EQUAL(s1, s2)`
Macro to test if two strings are equal.
- #define `TEST_ASSERT_STR_NOTEQUAL(s1, s2)`
Macro to test if two strings are not equal.

Functions

- void `tests_assert_true` (const bool success, const char *suiteName, const char *caseName, const char *failMessage, const char *fileName, const int lineNumber)
Logs the result of a true/false unit test.
- bool `tests_assert_almost_equal` (const long double a, const long double b, const long double e)
Tests two real numbers for fuzzy equality.
- void `tests_initialize` (void)
Initializes the test runner.
- void `tests_report` (void)
Reports on the test results.
- int `tests_get_total_tests` (void)
Returns the total number of tests run.
- int `tests_get_successes` (void)
Returns the total number of successful tests.
- int `tests_get_failures` (void)
Returns the total number of failed tests.

8.29.1 Detailed Description

Interface to unit test logging functionality.

Author

Paul Griffiths

Copyright

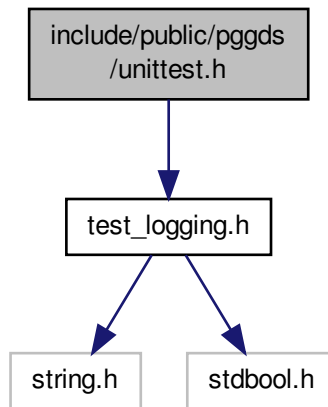
Copyright 2014 Paul Griffiths. Distributed under the terms of the GNU General Public License. <http://www.gnu.org/licenses/>

8.30 include/public/pggds/unittest.h File Reference

Public interface to unit test functionality.

```
#include "test_logging.h"
```

Include dependency graph for unittest.h:



8.30.1 Detailed Description

Public interface to unit test functionality.

Author

Paul Griffiths

Copyright

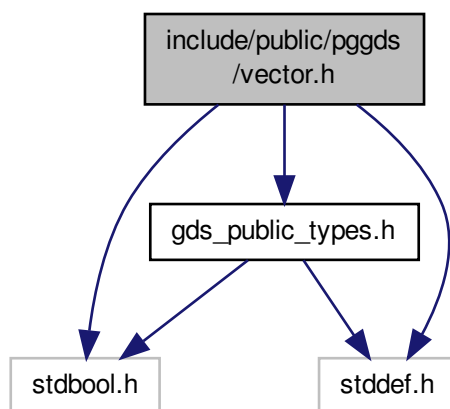
Copyright 2014 Paul Griffiths. Distributed under the terms of the GNU General Public License. <http://www.gnu.org/licenses/>

8.31 include/public/pggds/vector.h File Reference

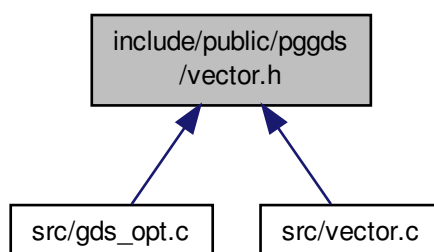
Interface to generic vector data structure.

```
#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.

8.31.1 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/>

8.32 src/dict.c File Reference

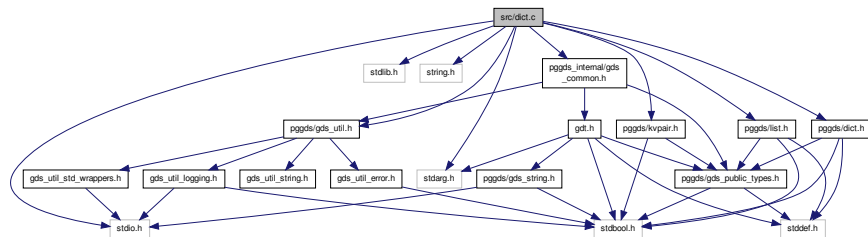
Implementation of generic dictionary data structure.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
#include <pggds_internal/gds_common.h>
#include <pggds/gds_util.h>
#include <pggds/dict.h>
#include <pggds/list.h>
#include <pggds/kvpair.h>

```

Include dependency graph for dict.c:



Data Structures

- struct [dict](#)

Functions

- 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

8.32.1 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/>

8.32.2 Function Documentation

8.32.2.1 `static bool dict_buckets_create (Dict dict) [static]`

Helper function to create the dictionary buckets.

Parameters

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

Return values

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

8.32.2.2 `static void dict_buckets_destroy (Dict dict) [static]`

Helper function to destroy the dictionary buckets.

Parameters

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

8.32.2.3 `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.

8.32.2.4 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.
-------------	------------------------------

8.32.2.5 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

8.32.2.6 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

8.32.2.7 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.
<i>...</i>	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

8.32.2.8 `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

8.32.2.9 `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

8.32.3 Variable Documentation

8.32.3.1 `const size_t BUCKETS = 256 [static]`

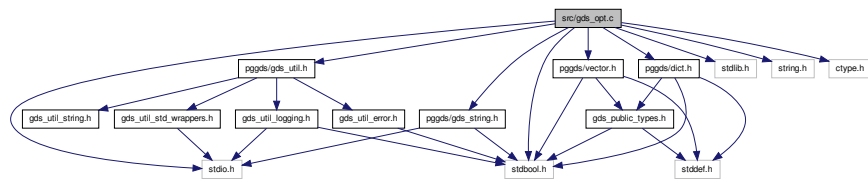
Number of buckets

8.33 `src/gds_opt.c` File Reference

Implementation of command line option functions.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <ctype.h>
#include <pggds/gds_util.h>
#include <pggds/dict.h>
#include <pggds/vector.h>
#include <pggds/gds_string.h>
```

Include dependency graph for gds_opt.c:



Macros

- `#define` [GDSDEBUG](#)

Enumerations

- `enum` [gds_argument_type](#) { [GDS_ARGUMENT_NO](#), [GDS_ARGUMENT_YES](#) }

Functions

- static [Dict](#) [gds_get_recognized_options](#) (const char *allowed)
Returns a dictionary of recognized options.
- static bool [create_static_structures](#) (void)
Creates the static structures used by the module.
- static void [destroy_static_structures](#) (void)
Destroys the static structures used by the module.
- bool [gds_parse_options](#) (const char *allowed, char **argv)
Parses a command line for options and non-options.
- void [gds_free_options](#) (void)
Frees memory associated with command line options.
- const char * [gds_option_progname](#) (void)
Returns the program name.
- bool [gds_option_present](#) (const char *optname)
Checks if an option was provided on the command line.
- const char * [gds_option_argument_string](#) (const char *optname)
Retrieves a string argument for an option.
- bool [gds_option_argument_int](#) (const char *optname, int *value)
Retrieves an integer argument for an option.
- int [gds_option_nonopts_number](#) (void)
Returns the number of non-option arguments provided.
- const char * [gds_option_nonopt](#) (const size_t index)
Retrieves a non-option argument.

Variables

- static const char * [progname](#) = NULL
- static bool [parsed](#) = false
- static [Dict](#) [options](#) = NULL
- static [Vector](#) [nonopts](#) = NULL

8.33.1 Detailed Description

Implementation of command line option functions.

Author

Paul Griffiths

Copyright

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

8.33.2 Macro Definition Documentation

8.33.2.1 #define GDSDEBUG

8.33.3 Enumeration Type Documentation

8.33.3.1 enum gds_argument_type

Enumeration type for option argument

Enumerator:

GDS_ARGUMENT_NO Option does not take an argument

GDS_ARGUMENT_YES Option takes an argument

8.33.4 Function Documentation

8.33.4.1 static bool create_static_structures (void) [static]

Creates the static structures used by the module.

8.33.4.2 static void destroy_static_structures (void) [static]

Destroys the static structures used by the module.

8.33.4.3 static Dict gds_get_recognized_options (const char * *allowed*) [static]

Returns a dictionary of recognized options.

The dictionary is created to match the provided string.

Parameters

<i>allowed</i>	A string representation of the allowed options. Each option should be represented by a single alphabetic character. Each option can optionally be followed by a ':' to show that it can accept an argument.
----------------	---

Returns

A dictionary containing the recognized options.

8.33.5 Variable Documentation

8.33.5.1 Vector nonopts = NULL [static]

File scope vector to hold non-option arguments

8.33.5.2 Dict options = NULL [static]

File scope dictionary to hold parsed options

8.33.5.3 bool parsed = false [static]

File scope variable to signify if command line has been parsed

8.33.5.4 const char* progname = NULL [static]

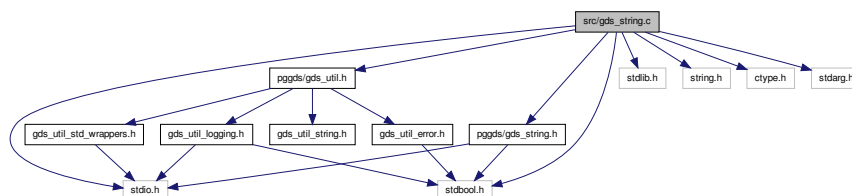
File scope variable to hold program name

8.34 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 <pggds/gds_string.h>
#include <pggds/gds_util.h>
```

Include dependency graph for gds_string.c:



Data Structures

- struct [GDSSString](#)

Functions

- static [GDSString gds_str_assign_cstr_direct](#) ([GDSString](#) dst, char *src, const size_t size, const size_t length)
Directly assigns dynamically allocated data to a string.
- static [GDSString gds_str_assign_cstr_length](#) ([GDSString](#) 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](#) ([GDSString](#) str, const size_t new_capacity)
Changes the capacity of a string.
- static bool [change_capacity_if_needed](#) ([GDSString](#) str, const size_t required_capacity)
Changes the capacity of a string if needed.
- static void [truncate_if_needed](#) ([GDSString](#) str)
Truncates a string if necessary.
- static [GDSString gds_str_concat_cstr_size](#) ([GDSString](#) 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](#) ([GDSString](#) str, const size_t numchars)
Removes characters at the start of a string, in place.
- static void [gds_str_remove_right](#) ([GDSString](#) str, const size_t numchars)
Removes characters at the end of a string, in place.
- [GDSString gds_str_create_direct](#) (char *init_str, const size_t init_str_size)
Creates a string using allocated memory.
- [GDSString gds_str_create](#) (const char *init_str)
Creates a new string from a C-style string.
- [GDSString gds_str_dup](#) ([GDSString](#) src)
Creates a new string from another string.
- [GDSString gds_str_create_sprintf](#) (const char *format,...)
Creates a string with `sprintf()`-type format.
- void [gds_str_destroy](#) ([GDSString](#) 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` (GDSSString s1, const char *s2)

Compares a string with a C-style string.
- int `gds_str_strchr` (GDSSString str, const char ch, const int start)

Returns index of first occurrence of a character.
- GDSSString `gds_str_substr_left` (GDSSString str, const size_t numchars)

Returns a left substring.
- GDSSString `gds_str_substr_right` (GDSSString str, const size_t numchars)

Returns a right substring.
- void `gds_str_split` (GDSSString src, GDSSString *left, GDSSString *right, const char sc)

Splits a string.
- void `gds_str_trim_leading` (GDSSString str)

Trims leading whitespace in-place.
- void `gds_str_trim_trailing` (GDSSString str)

Trims trailing whitespace in-place.
- void `gds_str_trim` (GDSSString str)

Trims leading and trailing whitespace in-place.
- char `gds_str_char_at_index` (GDSSString str, const size_t index)

Returns the character at a specified index.
- bool `gds_str_is_empty` (GDSSString str)

Checks if a string is empty.
- bool `gds_str_is_alnum` (GDSSString str)

Checks if a string contains only alphanumeric characters.
- void `gds_str_clear` (GDSSString str)

Clears (empties) a string.
- bool `gds_str_intval` (GDSSString str, const int base, int *value)

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

Gets the double value of a string.
- GDSSString `gds_str_getline_assign` (GDSSString str, const size_t size, FILE *fp)

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

Gets a line from a file creates a new string.
- GDSSString `gds_str_decorate` (GDSSString str, GDSSString left_dec, GDSSString right_dec)

Brackets a string with decoration strings.

8.34.1 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/>

8.34.2 Function Documentation

8.34.2.1 static bool change_capacity (GDSString *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.

8.34.2.2 static bool change_capacity_if_needed (GDSString *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.

8.34.2.3 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.

8.34.2.4 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.

8.34.2.5 `static GDSString gds_str_assign_ctr_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.

8.34.2.6 `static GDSString gds_str_concat_ctr_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.

8.34.2.7 `void gds_str_destructor (void * str)`

8.34.2.8 `static void gds_str_remove_left (GDSString 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.

8.34.2.9 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.

8.34.2.10 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.

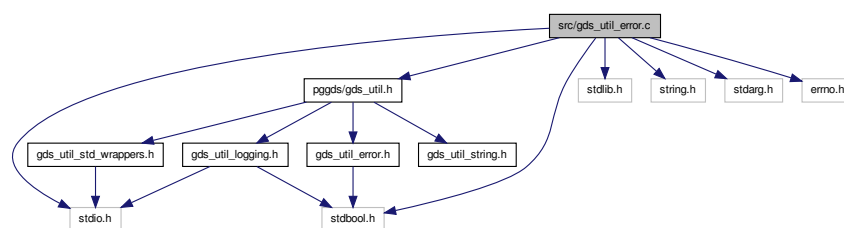
Parameters

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

8.35 src/gds_util_error.c File Reference

Implementation of general utility error functions.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <stdarg.h>
#include <errno.h>
#include <pggds/gds_util.h>
Include dependency graph for gds_util_error.c:
```



Functions

- void [gds_logerror_line](#) (const char *[progrname](#), const char *[filename](#), const int [linenum](#), const bool [log_erno](#), const enum [gds_error_quit_type](#) [quit_type](#), const char *[fmt](#),...)
Logs an error message.

8.35.1 Detailed Description

Implementation of general utility error functions.

Author

Paul Griffiths

Copyright

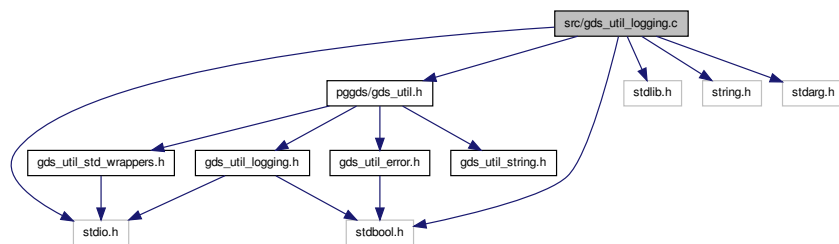
Copyright 2014 Paul Griffiths. Distributed under the terms of the GNU General Public License. <http://www.gnu.org/licenses/>

8.36 src/gds_util_logging.c File Reference

Implementation of logging functions.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <stdarg.h>
#include <pggds/gds_util.h>
```

Include dependency graph for gds_util_logging.c:

**Functions**

- FILE * [gds_errlog](#) (void)
Returns a pointer to the current log file.
- bool [gds_logging_on](#) (const char *logfile, const bool append)
Starts logging functionality.
- bool [gds_logging_off](#) (void)
Stops logging functionality.
- void [gds_log_msg](#) (const char *fmt,...)

Variables

- static FILE * [gds_error_file](#) = NULL
- static char * [gds_error_file_name](#) = NULL
- static bool [gds_logging_enabled](#) = false

8.36.1 Detailed Description

Implementation of logging functions.

Author

Paul Griffiths

Copyright

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

8.36.2 Function Documentation

8.36.2.1 `void gds_log_msg (const char * fmt, ...)`

8.36.3 Variable Documentation

8.36.3.1 `FILE* gds_error_file = NULL` `[static]`

File scope variable to hold current error file pointer

8.36.3.2 `char* gds_error_file_name = NULL` `[static]`

File scope variable to hold current error file name

8.36.3.3 `bool gds_logging_enabled = false` `[static]`

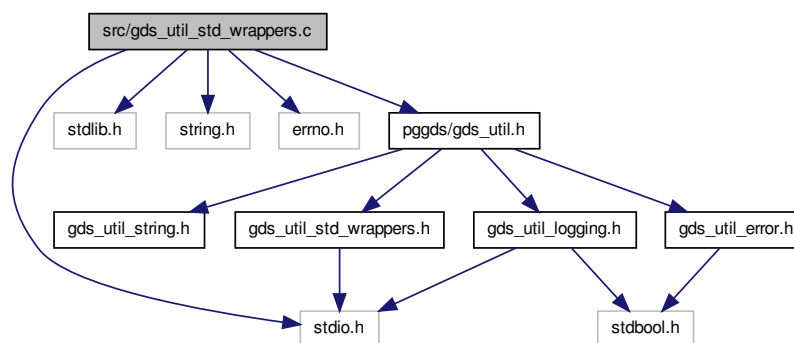
File scope variable for current logging status

8.37 src/gds_util_std_wrappers.c File Reference

Implementation of wrappers for standard functions.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <ppgds/gds_util.h>
```

Include dependency graph for gds_util_std_wrappers.c:



Functions

- void * [gds_xmalloc](#) (const size_t size, const char *filename, const int linenum)
Wraps malloc() and aborts on failure.
- void * [gds_xcalloc](#) (const size_t nmemb, const size_t size, const char *filename, const int linenum)
Wraps calloc() and aborts on failure.
- void * [gds_xrealloc](#) (void *ptr, const size_t size, const char *filename, const int linenum)
Wraps realloc() and aborts on failure.
- char * [gds_xstrdup](#) (const char *str, const char *filename, const int linenum)
Wraps strdup() and aborts on failure.
- FILE * [gds_xfopen](#) (const char *path, const char *mode, const char *filename, const int linenum)
Wraps fopen() and exits on failure.

8.37.1 Detailed Description

Implementation of wrappers for standard functions.

Author

Paul Griffiths

Copyright

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

8.37.2 Function Documentation

8.37.2.1 void* gds_xcalloc (const size_t nmemb, const size_t size, const char * filename, const int linenum)

Wraps calloc() and aborts on failure.

This is designed to be called from the corresponding macro.

Parameters

<i>nmemb</i>	The number of members to allocate.
<i>size</i>	The size in bytes of each member.
<i>filename</i>	The name of the calling file.
<i>linenum</i>	The line number in the calling file.

Returns

A pointer to the allocated memory.

8.37.2.2 FILE* gds_xfopen (const char * path, const char * mode, const char * filename, const int linenum)

Wraps fopen() and exits on failure.

This is designed to be called from the corresponding macro.

Parameters

<i>path</i>	The path of the file to open.
<i>mode</i>	The mode under which to open the file.
<i>filename</i>	The name of the calling file.
<i>linenum</i>	The line number in the calling file.

Returns

A pointer to the allocated memory.

8.37.2.3 void* gds_xmalloc (const size_t *size*, const char * *filename*, const int *linenum*)

Wraps malloc() and aborts on failure.

This is designed to be called from the corresponding macro.

Parameters

<i>size</i>	The number of bytes to allocate.
<i>filename</i>	The name of the calling file.
<i>linenum</i>	The line number in the calling file.

Returns

A pointer to the allocated memory.

8.37.2.4 void* gds_xrealloc (void * *ptr*, const size_t *size*, const char * *filename*, const int *linenum*)

Wraps realloc() and aborts on failure.

This is designed to be called from the corresponding macro.

Parameters

<i>ptr</i>	A pointer to the memory to reallocate.
<i>size</i>	The number of bytes for the new allocation.
<i>filename</i>	The name of the calling file.
<i>linenum</i>	The line number in the calling file.

Returns

A pointer to the reallocated memory.

8.37.2.5 char* gds_xstrdup (const char * *str*, const char * *filename*, const int *linenum*)

Wraps strdup() and aborts on failure.

This is designed to be called from the corresponding macro.

Parameters

<i>str</i>	The string to duplicate.
<i>filename</i>	The name of the calling file.
<i>linenum</i>	The line number in the calling file.

Returns

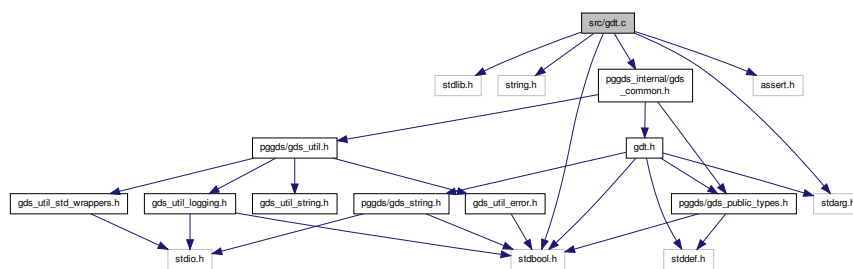
A pointer to the allocated memory.

8.38 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 <pggds_internal/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.

- static int `gdt_compare_gds_str` (const void *p1, const void *p2)

Compare function for `GDSString`.

- 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.

8.38.1 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/>

8.38.2 Function Documentation

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

Compare function for char.

Parameters

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

Return values

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

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

Compare function for double.

Parameters

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

Return values

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

8.38.2.3 `static int gdt_compare_gds_str (const void * p1, const void * p2)` `[static]`

Compare function for [GDString](#).

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

8.38.2.4 `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

8.38.2.5 `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

8.38.2.6 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

8.38.2.7 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

8.38.2.8 static int gdt_compare_sizet (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

8.38.2.9 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

8.38.2.10 `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

8.38.2.11 `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

8.38.2.12 `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

8.38.2.13 `static int gds_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

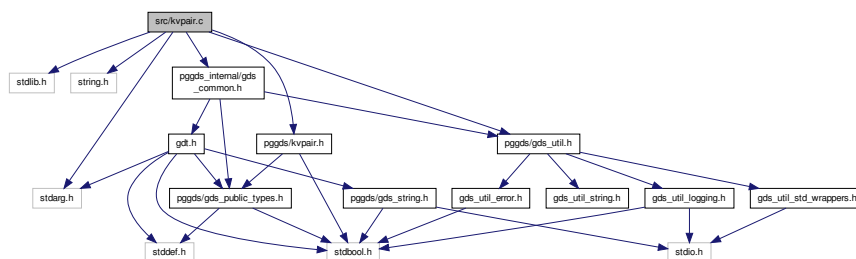
<i>0</i>	First value is equal to second value
<i>-1</i>	First value is less than second value
<i>1</i>	First value is greater than second value

8.39 src/kvpair.c File Reference

Implementation of generic key-value pair structure.

```
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
#include <pggds_internal/gds_common.h>
#include <pggds/gds_util.h>
#include <pggds/kvpair.h>
```

Include dependency graph for kvpair.c:



Functions

- [KVPair gds_kvpair_create](#) (const char *key, const enum [gds_datatype](#) type, va_list ap)
Creates a new key-value pair.
- void [gds_kvpair_destroy](#) (KVPair pair, const bool free_value)
Destroys a key-value pair.
- int [gds_kvpair_compare](#) (const void *p1, const void *p2)
Compares two key-value pairs by key.

8.39.1 Detailed Description

Implementation of generic key-value pair structure.

Author

Paul Griffiths

Copyright

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

8.39.2 Function Documentation

8.39.2.1 `int gds_kvpair_compare (const void * p1, const void * p2)`

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

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

8.39.2.2 `KVPair gds_kvpair_create (const char * key, const enum gds_datatype type, va_list ap)`

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 <code>va_list</code> containing the data value for the pair. 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>	Success

8.39.2.3 `void gds_kvpair_destroy (KVPair pair, const bool free_value)`

Destroys a key-value pair.

Parameters

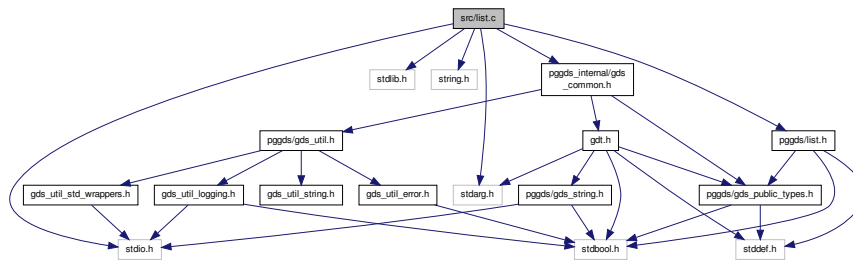
<i>pair</i>	A pointer to the pair to destroy.
<i>free_value</i>	If true, the data will be passed to gdt_free()

8.40 src/list.c File Reference

Implementation of generic list data structure.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
#include <pggds_internal/gds_common.h>
#include <pggds/list.h>
```

Include dependency graph for list.c:



Data Structures

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

Typedefs

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

Functions

- static [ListNode](#) [list_node_create](#) ([List](#) list, va_list ap)
Private function to create list node.
- static void [list_node_destroy](#) ([List](#) list, [ListNode](#) node)
Destroys a list node.
- static [ListNode](#) [list_node_at_index](#) ([List](#) list, const size_t index)
Private function to return the node at a specified index.
- static bool [list_insert_internal](#) ([List](#) list, [ListNode](#) node, const size_t index)
Private function to insert a node into a list.
- [List](#) [list_create](#) (const enum [gds_datatype](#) type, const int opts,...)
Creates a new list.
- void [list_destroy](#) ([List](#) list)
Destroys a list.
- bool [list_append](#) ([List](#) list,...)
Appends a value to the back of a list.
- bool [list_prepend](#) ([List](#) list,...)
Prepends a value to the front of a list.
- bool [list_insert](#) ([List](#) list, const size_t index,...)
Inserts a value into a list.

- bool `list_delete_index` (`List list`, const `size_t index`)
Deletes the value at the specified index of the list.
- bool `list_delete_front` (`List list`)
Deletes the value at the front of the list.
- bool `list_delete_back` (`List list`)
Deletes the value at the back of the list.
- bool `list_element_at_index` (`List list`, const `size_t index`, void *`p`)
Gets the value at the specified index of the list.
- bool `list_set_element_at_index` (`List list`, const `size_t index`,...)
Sets the value at the specified index of the list.
- bool `list_find` (`List list`, `size_t *index`,...)
Tests if a value is contained in a list.
- `ListItr list_find_itr` (`List list`,...)
Tests if a value is contained in a list.
- bool `list_sort` (`List list`)
Sorts a list in-place, in ascending order.
- bool `list_reverse_sort` (`List list`)
Sorts a list in-place, in descending order.
- `ListItr list_itr_first` (`List list`)
Returns an iterator to the first element of the list.
- `ListItr list_itr_last` (`List list`)
Returns an iterator to the last element of the list.
- `ListItr list_itr_next` (`ListItr itr`)
Increments a list iterator.
- `ListItr list_itr_previous` (`ListItr itr`)
Decrements a list iterator.
- void `list_get_value_itr` (`ListItr itr`, void *`p`)
Retrieves a value from an iterator.
- bool `list_is_empty` (`List list`)
Tests if a list is empty.
- `size_t list_length` (`List list`)
Returns the length of a list.

8.40.1 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/>

8.40.2 Typedef Documentation

8.40.2.1 typedef struct list_node * ListNode

List node structure

8.40.3 Function Documentation

8.40.3.1 `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

8.40.3.2 `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

8.40.3.3 `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 <code>va_list</code> 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

8.40.3.4 `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

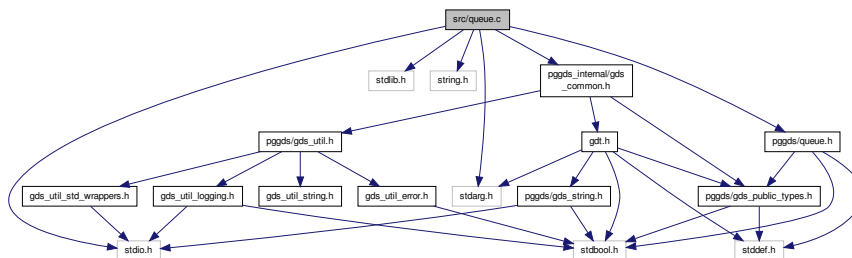
<i>list</i>	A pointer to the list.
<i>node</i>	A pointer to the node.

8.41 src/queue.c File Reference

Implementation of generic queue data structure.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
#include <ppgds_internal/gds_common.h>
#include <ppgds/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.

8.41.1 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/>

8.41.2 Variable Documentation

8.41.2.1 const size_t GROWTH = 2 [static]

Growth factor for dynamic memory allocation.

Attention

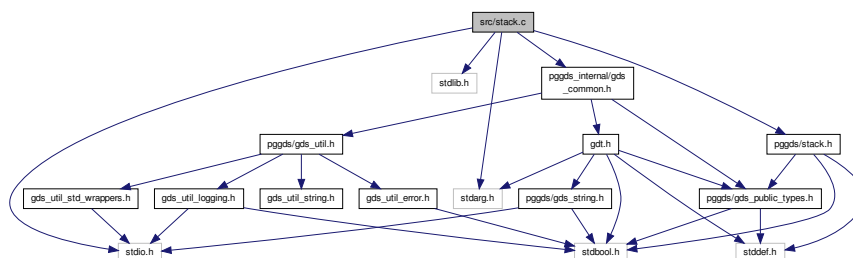
`queue_push()` relies on this being at least 2.

8.42 src/stack.c File Reference

Implementation of generic stack data structure.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <ppgds_internal/gds_common.h>
#include <ppgds/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

8.42.1 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/>

8.42.2 Variable Documentation

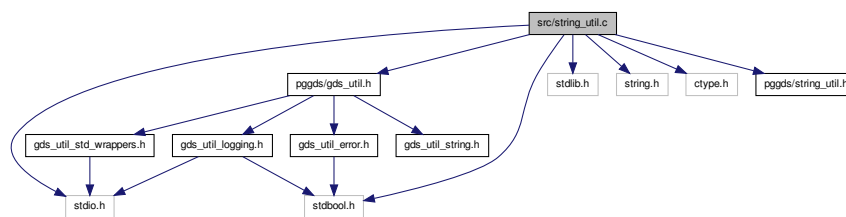
8.42.2.1 const size_t GROWTH = 2 [static]

Growth factor for dynamic memory allocation

8.43 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 <pggds/gds_util.h>
#include <pggds/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.

8.43.1 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/>

8.43.2 Function Documentation

8.43.2.1 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.

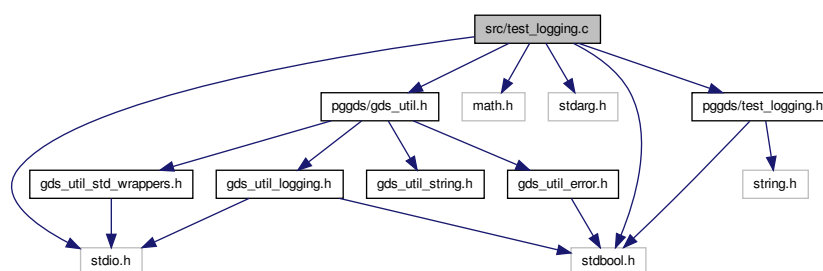
Return values

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

8.44 src/test_logging.c File Reference

Implementation of unit test logging functionality.

```
#include <stdio.h>
#include <stdbool.h>
#include <math.h>
#include <stdarg.h>
#include <ppgds/gds_util.h>
#include <ppgds/test_logging.h>
Include dependency graph for test_logging.c:
```



Functions

- static void `tests_log_single_test` (const bool success)
Logs the result of a single test.
- void `tests_assert_true` (const bool success, const char *suiteName, const char *caseName, const char *failMessage, const char *fileName, const int lineNumber)
Logs the result of a true/false unit test.
- bool `tests_assert_almost_equal` (const long double a, const long double b, const long double e)
Tests two real numbers for fuzzy equality.
- void `tests_initialize` (void)
Initializes the test runner.
- void `tests_report` (void)
Reports on the test results.
- int `tests_get_total_tests` (void)
Returns the total number of tests run.
- int `tests_get_successes` (void)
Returns the total number of successful tests.
- int `tests_get_failures` (void)
Returns the total number of failed tests.

Variables

- static int `test_successes` = 0
- static int `test_failures` = 0
- static int `total_tests` = 0
- static bool `show_failures` = true

8.44.1 Detailed Description

Implementation of unit test logging functionality.

Author

Paul Griffiths

Copyright

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

8.44.2 Function Documentation

8.44.2.1 static void `tests_log_single_test` (const bool *success*) [static]

Logs the result of a single test.

Parameters

<i>success</i>	true if the test passed, false if it failed.
----------------	--

8.44.3 Variable Documentation

8.44.3.1 `bool show_failures = true` `[static]`

Control flag to display individual test failures

8.44.3.2 `int test_failures = 0` `[static]`

Number of failed tests

8.44.3.3 `int test_successes = 0` `[static]`

Number of successful tests

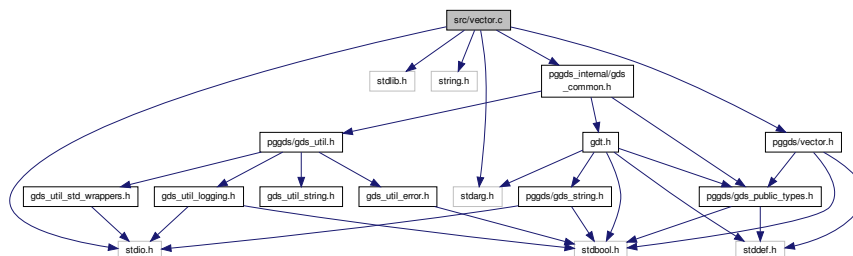
8.44.3.4 `int total_tests = 0` `[static]`

Total number of tests

8.45 src/vector.c File Reference

Implementation of generic vector data structure.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
#include <pggds_internal/gds_common.h>
#include <pggds/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

8.45.1 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/>

8.45.2 Function Documentation

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

Private function to insert a vector element.

Parameters

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

Return values

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

8.45.3 Variable Documentation

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

Growth factor for dynamic memory allocation

Index

- abort_error
 - Public general generic data structures functionality, [29](#)
- BUCKETS
 - dict.c, [118](#)
- back
 - queue, [75](#)
- buckets
 - dict, [66](#)
- c
 - gdt_generic_datatype, [69](#)
- capacity
 - GDSString, [68](#)
 - queue, [75](#)
 - stack, [77](#)
 - vector, [79](#)
- change_capacity
 - gds_string.c, [124](#)
- change_capacity_if_needed
 - gds_string.c, [124](#)
- compfunc
 - gdt_generic_datatype, [69](#)
 - list, [71](#)
 - vector, [79](#)
- create_static_structures
 - gds_opt.c, [120](#)
- d
 - gdt_generic_datatype, [69](#)
- DATATYPE_CHAR
 - Private functionality for manipulating generic datatypes, [25](#)
- DATATYPE_DOUBLE
 - Private functionality for manipulating generic datatypes, [25](#)
- DATATYPE_GDSSTRING
 - Private functionality for manipulating generic datatypes, [25](#)
- DATATYPE_INT
 - Private functionality for manipulating generic datatypes, [25](#)
- DATATYPE_LONG
 - Private functionality for manipulating generic datatypes, [25](#)
- DATATYPE_LONG_LONG
 - Private functionality for manipulating generic datatypes, [25](#)
- DATATYPE_POINTER
 - Private functionality for manipulating generic datatypes, [25](#)
- DATATYPE_SIGNED_CHAR
 - Private functionality for manipulating generic datatypes, [25](#)
- DATATYPE_SIZE_T
 - Private functionality for manipulating generic datatypes, [25](#)
- DATATYPE_STRING
 - Private functionality for manipulating generic datatypes, [25](#)
- DATATYPE_UNSIGNED_CHAR
 - Private functionality for manipulating generic datatypes, [25](#)
- DATATYPE_UNSIGNED_INT
 - Private functionality for manipulating generic datatypes, [25](#)
- DATATYPE_UNSIGNED_LONG
 - Private functionality for manipulating generic datatypes, [25](#)
- DATATYPE_UNSIGNED_LONG_LONG
 - Private functionality for manipulating generic datatypes, [25](#)
- DPRINTF
 - gds_util_logging.h, [96](#)
- data
 - GDSString, [68](#)
 - gdt_generic_datatype, [69](#)
- destroy_static_structures
 - gds_opt.c, [120](#)
- Dict
 - dict.h, [85](#)
- dict, [65](#)
 - buckets, [66](#)
 - exit_on_error, [66](#)
 - free_on_destroy, [66](#)
 - num_buckets, [66](#)
 - type, [66](#)
- dict.c
 - BUCKETS, [118](#)
 - dict_buckets_create, [116](#)
 - dict_buckets_destroy, [116](#)
 - dict_create, [116](#)
 - dict_destroy, [116](#)
 - dict_has_key, [117](#)
 - dict_has_key_internal, [117](#)
 - dict_insert, [117](#)
 - dict_value_for_key, [118](#)
 - djb2hash, [118](#)

- dict.h
 - Dict, [85](#)
 - dict_create, [86](#)
 - dict_destroy, [86](#)
 - dict_has_key, [86](#)
 - dict_insert, [86](#)
 - dict_value_for_key, [87](#)
- dict_buckets_create
 - dict.c, [116](#)
- dict_buckets_destroy
 - dict.c, [116](#)
- dict_create
 - dict.c, [116](#)
 - dict.h, [86](#)
- dict_destroy
 - dict.c, [116](#)
 - dict.h, [86](#)
- dict_has_key
 - dict.c, [117](#)
 - dict.h, [86](#)
- dict_has_key_internal
 - dict.c, [117](#)
- dict_insert
 - dict.c, [117](#)
 - dict.h, [86](#)
- dict_value_for_key
 - dict.c, [118](#)
 - dict.h, [87](#)
- djb2hash
 - dict.c, [118](#)
- docs/cmdline.dox, [81](#)
- docs/gds.dox, [81](#)
- docs/gds_string.dox, [81](#)
- docs/gdt.dox, [81](#)
- docs/general.dox, [81](#)
- docs/list.dox, [81](#)
- docs/logging.dox, [81](#)
- docs/queue.dox, [81](#)
- docs/stack.dox, [81](#)
- docs/string_util.dox, [81](#)
- docs/unittest.dox, [81](#)
- docs/vector.dox, [81](#)
- duplicate_cstr
 - gds_string.c, [124](#)
- element
 - list_node, [73](#)
- elements
 - queue, [75](#)
 - stack, [77](#)
 - vector, [79](#)
- exit_on_error
 - dict, [66](#)
 - list, [71](#)
 - queue, [76](#)
 - stack, [77](#)
 - vector, [79](#)
- first
 - pair_string, [74](#)
- free_on_destroy
 - dict, [66](#)
 - list, [72](#)
 - queue, [76](#)
 - stack, [77](#)
 - vector, [79](#)
- front
 - queue, [76](#)
- GDS_ARGUMENT_NO
 - gds_opt.c, [120](#)
- GDS_ARGUMENT_YES
 - gds_opt.c, [120](#)
- GDS_ERROR_ABORT
 - gds_util_error.h, [94](#)
- GDS_ERROR_ASSERT
 - gds_util_error.h, [94](#)
- GDS_ERROR_EXIT
 - gds_util_error.h, [94](#)
- GDS_ERROR_NOQUIT
 - gds_util_error.h, [94](#)
- GDS_EXIT_ON_ERROR
 - Public general generic data structures functionality, [31](#)
- GDS_FREE_ON_DESTROY
 - Public general generic data structures functionality, [31](#)
- GDS_RESIZABLE
 - Public general generic data structures functionality, [31](#)
- GDSDEBUG
 - gds_opt.c, [120](#)
- GDSString, [67](#)
 - capacity, [68](#)
 - data, [68](#)
 - length, [68](#)
 - Public interface to string data structure, [15](#)
- GDSString_destructor
 - Public interface to string data structure, [23](#)
- GROWTH
 - queue.c, [142](#)
 - stack.c, [143](#)
 - vector.c, [149](#)
- gds_opt.c
 - GDS_ARGUMENT_NO, [120](#)
 - GDS_ARGUMENT_YES, [120](#)
- gds_util_error.h
 - GDS_ERROR_ABORT, [94](#)
 - GDS_ERROR_ASSERT, [94](#)
 - GDS_ERROR_EXIT, [94](#)
 - GDS_ERROR_NOQUIT, [94](#)
- gds_argument_type
 - gds_opt.c, [120](#)
- gds_assert
 - Public general generic data structures functionality, [29](#)
- gds_cfunc

- Private functionality for manipulating generic datatypes, [24](#)
- `gds_datatype`
 - Private functionality for manipulating generic datatypes, [25](#)
- `gds_errlog`
 - Public interface to logging functionality, [40](#)
- `gds_error_file`
 - `gds_util_logging.c`, [128](#)
- `gds_error_file_name`
 - `gds_util_logging.c`, [128](#)
- `gds_error_quit_type`
 - `gds_util_error.h`, [94](#)
- `gds_free_options`
 - Public interface to command line parsing functionality, [11](#)
- `gds_get_recognized_options`
 - `gds_opt.c`, [120](#)
- `gds_kvpair`, [66](#)
 - key, [67](#)
 - value, [67](#)
- `gds_kvpair_compare`
 - `kvpair.c`, [137](#)
 - `kvpair.h`, [101](#)
- `gds_kvpair_create`
 - `kvpair.c`, [137](#)
 - `kvpair.h`, [101](#)
- `gds_kvpair_destroy`
 - `kvpair.c`, [137](#)
 - `kvpair.h`, [102](#)
- `gds_log_msg`
 - `gds_util_logging.c`, [128](#)
 - `gds_util_logging.h`, [96](#)
- `gds_logerror_line`
 - Public general generic data structures functionality, [31](#)
- `gds_logging_enabled`
 - `gds_util_logging.c`, [128](#)
- `gds_logging_off`
 - Public interface to logging functionality, [40](#)
- `gds_logging_on`
 - Public interface to logging functionality, [40](#)
- `gds_opt.c`
 - `create_static_structures`, [120](#)
 - `destroy_static_structures`, [120](#)
 - `GDSDEBUG`, [120](#)
 - `gds_argument_type`, [120](#)
 - `gds_get_recognized_options`, [120](#)
 - `nonopts`, [121](#)
 - `options`, [121](#)
 - `parsed`, [121](#)
 - `prognome`, [121](#)
- `gds_option`
 - Public general generic data structures functionality, [31](#)
- `gds_option_argument_int`
 - Public interface to command line parsing functionality, [11](#)
- `gds_option_argument_string`
 - Public interface to command line parsing functionality, [12](#)
- `gds_option_nonopt`
 - Public interface to command line parsing functionality, [12](#)
- `gds_option_nonopts_number`
 - Public interface to command line parsing functionality, [12](#)
- `gds_option_present`
 - Public interface to command line parsing functionality, [12](#)
- `gds_option_prognome`
 - Public interface to command line parsing functionality, [13](#)
- `gds_parse_options`
 - Public interface to command line parsing functionality, [13](#)
- `gds_str_assign`
 - Public interface to string data structure, [15](#)
- `gds_str_assign_cstr`
 - Public interface to string data structure, [15](#)
- `gds_str_assign_cstr_direct`
 - `gds_string.c`, [124](#)
- `gds_str_assign_cstr_length`
 - `gds_string.c`, [125](#)
- `gds_str_char_at_index`
 - Public interface to string data structure, [16](#)
- `gds_str_clear`
 - Public interface to string data structure, [16](#)
- `gds_str_compare`
 - Public interface to string data structure, [16](#)
- `gds_str_compare_cstr`
 - Public interface to string data structure, [16](#)
- `gds_str_concat`
 - Public interface to string data structure, [17](#)
- `gds_str_concat_cstr`
 - Public interface to string data structure, [17](#)
- `gds_str_concat_cstr_size`
 - `gds_string.c`, [125](#)
- `gds_str_create`
 - Public interface to string data structure, [17](#)
- `gds_str_create_direct`
 - Public interface to string data structure, [17](#)
- `gds_str_create_sprintf`
 - Public interface to string data structure, [18](#)
- `gds_str_cstr`
 - Public interface to string data structure, [18](#)
- `gds_str_decorate`
 - Public interface to string data structure, [18](#)
- `gds_str_destroy`
 - Public interface to string data structure, [18](#)
- `gds_str_destructor`
 - `gds_string.c`, [125](#)
- `gds_str_doubleval`
 - Public interface to string data structure, [19](#)
- `gds_str_dup`
 - Public interface to string data structure, [19](#)

- gds_str_getline
 - Public interface to string data structure, [19](#)
- gds_str_getline_assign
 - Public interface to string data structure, [19](#)
- gds_str_hash
 - Public interface to string data structure, [20](#)
- gds_str_intval
 - Public interface to string data structure, [20](#)
- gds_str_is_alnum
 - Public interface to string data structure, [20](#)
- gds_str_is_empty
 - Public interface to string data structure, [20](#)
- gds_str_length
 - Public interface to string data structure, [21](#)
- gds_str_remove_left
 - gds_string.c, [125](#)
- gds_str_remove_right
 - gds_string.c, [125](#)
- gds_str_size_to_fit
 - Public interface to string data structure, [21](#)
- gds_str_split
 - Public interface to string data structure, [21](#)
- gds_str_strchr
 - Public interface to string data structure, [21](#)
- gds_str_substr_left
 - Public interface to string data structure, [22](#)
- gds_str_substr_right
 - Public interface to string data structure, [22](#)
- gds_str_trim
 - Public interface to string data structure, [22](#)
- gds_str_trim_leading
 - Public interface to string data structure, [22](#)
- gds_str_trim_trailing
 - Public interface to string data structure, [22](#)
- gds_str_trunc
 - Public interface to string data structure, [23](#)
- gds_strdup
 - General purpose string manipulation functions, [49](#)
 - Public general generic data structures functionality, [32](#)
- gds_string.c
 - change_capacity, [124](#)
 - change_capacity_if_needed, [124](#)
 - duplicate_cstr, [124](#)
 - gds_str_assign_cstr_direct, [124](#)
 - gds_str_assign_cstr_length, [125](#)
 - gds_str_concat_cstr_size, [125](#)
 - gds_str_destructor, [125](#)
 - gds_str_remove_left, [125](#)
 - gds_str_remove_right, [125](#)
 - truncate_if_needed, [126](#)
- gds_strndup
 - General purpose string manipulation functions, [50](#)
- gds_trim
 - General purpose string manipulation functions, [50](#)
- gds_trim_left
 - General purpose string manipulation functions, [50](#)
- gds_trim_line_ending
 - General purpose string manipulation functions, [50](#)
- gds_trim_right
 - General purpose string manipulation functions, [51](#)
- gds_util_error.h
 - gds_error_quit_type, [94](#)
- gds_util_logging.c
 - gds_error_file, [128](#)
 - gds_error_file_name, [128](#)
 - gds_log_msg, [128](#)
 - gds_logging_enabled, [128](#)
- gds_util_logging.h
 - DPRINTF, [96](#)
 - gds_log_msg, [96](#)
- gds_util_std_wrappers.c
 - gds_xcalloc, [129](#)
 - gds_xfopen, [129](#)
 - gds_xmalloc, [130](#)
 - gds_xrealloc, [130](#)
 - gds_xstrdup, [130](#)
- gds_util_std_wrappers.h
 - gds_xcalloc, [97](#)
 - gds_xfopen, [98](#)
 - gds_xmalloc, [98](#)
 - gds_xrealloc, [98](#)
 - gds_xstrdup, [98](#)
- gds_xcalloc
 - gds_util_std_wrappers.c, [129](#)
 - gds_util_std_wrappers.h, [97](#)
- gds_xfopen
 - gds_util_std_wrappers.c, [129](#)
 - gds_util_std_wrappers.h, [98](#)
- gds_xmalloc
 - gds_util_std_wrappers.c, [130](#)
 - gds_util_std_wrappers.h, [98](#)
- gds_xrealloc
 - gds_util_std_wrappers.c, [130](#)
 - gds_util_std_wrappers.h, [98](#)
- gds_xstrdup
 - gds_util_std_wrappers.c, [130](#)
 - gds_util_std_wrappers.h, [98](#)
- gdsstr
 - gdt_generic_datatype, [69](#)
- gdt.c
 - gdt_compare_char, [132](#)
 - gdt_compare_double, [132](#)
 - gdt_compare_gds_str, [133](#)
 - gdt_compare_int, [133](#)
 - gdt_compare_long, [133](#)
 - gdt_compare_longlong, [134](#)
 - gdt_compare_schar, [134](#)
 - gdt_compare_sizet, [134](#)
 - gdt_compare_string, [134](#)
 - gdt_compare_uchar, [135](#)
 - gdt_compare_uint, [135](#)
 - gdt_compare_ulong, [135](#)
 - gdt_compare_ulonglong, [135](#)
- gdt_compare

- Private functionality for manipulating generic datatypes, 25
- `gdt_compare_char`
 - `gdt.c`, 132
- `gdt_compare_double`
 - `gdt.c`, 132
- `gdt_compare_gds_str`
 - `gdt.c`, 133
- `gdt_compare_int`
 - `gdt.c`, 133
- `gdt_compare_long`
 - `gdt.c`, 133
- `gdt_compare_longlong`
 - `gdt.c`, 134
- `gdt_compare_schar`
 - `gdt.c`, 134
- `gdt_compare_sizet`
 - `gdt.c`, 134
- `gdt_compare_string`
 - `gdt.c`, 134
- `gdt_compare_uchar`
 - `gdt.c`, 135
- `gdt_compare_uint`
 - `gdt.c`, 135
- `gdt_compare_ulong`
 - `gdt.c`, 135
- `gdt_compare_ulonglong`
 - `gdt.c`, 135
- `gdt_compare_void`
 - Private functionality for manipulating generic datatypes, 25
- `gdt_free`
 - Private functionality for manipulating generic datatypes, 26
- `gdt_generic_datatype`, 68
 - `c`, 69
 - `compfunc`, 69
 - `d`, 69
 - `data`, 69
 - `gdsstr`, 69
 - `i`, 69
 - `l`, 69
 - `ll`, 69
 - `p`, 70
 - `pc`, 70
 - `sc`, 70
 - `st`, 70
 - `type`, 70
 - `uc`, 70
 - `ui`, 70
 - `ul`, 70
 - `ull`, 70
- `gdt_get_value`
 - Private functionality for manipulating generic datatypes, 26
- `gdt_reverse_compare_void`
 - Private functionality for manipulating generic datatypes, 26
- `gdt_set_value`
 - Private functionality for manipulating generic datatypes, 26
- General purpose string manipulation functions, 49
 - `gds_strdup`, 49
 - `gds_strndup`, 50
 - `gds_trim`, 50
 - `gds_trim_left`, 50
 - `gds_trim_line_ending`, 50
 - `gds_trim_right`, 51
 - `list_string_create`, 51
 - `list_string_destroy`, 51
 - `pair_string_copy`, 51
 - `pair_string_create`, 52
 - `pair_string_destroy`, 52
 - `split_string`, 52
- head
 - list, 72
- i
 - `gdt_generic_datatype`, 69
 - `include/private/pggds_internal/gds_common.h`, 81
 - `include/private/pggds_internal/gdt.h`, 82
 - `include/public/pggds/dict.h`, 84
 - `include/public/pggds/gds_opt.h`, 87
 - `include/public/pggds/gds_public_types.h`, 88
 - `include/public/pggds/gds_string.h`, 89
 - `include/public/pggds/gds_util.h`, 92
 - `include/public/pggds/gds_util_error.h`, 93
 - `include/public/pggds/gds_util_logging.h`, 94
 - `include/public/pggds/gds_util_std_wrappers.h`, 96
 - `include/public/pggds/gds_util_string.h`, 99
 - `include/public/pggds/kvpair.h`, 99
 - `include/public/pggds/list.h`, 102
 - `include/public/pggds/queue.h`, 104
 - `include/public/pggds/stack.h`, 106
 - `include/public/pggds/string_util.h`, 108
 - `include/public/pggds/test_logging.h`, 110
 - `include/public/pggds/unittest.h`, 111
 - `include/public/pggds/vector.h`, 112
- KVPair
 - `kvpair.h`, 101
- key
 - `gds_kvpair`, 67
- `kvpair.c`
 - `gds_kvpair_compare`, 137
 - `gds_kvpair_create`, 137
 - `gds_kvpair_destroy`, 137
- `kvpair.h`
 - `gds_kvpair_compare`, 101
 - `gds_kvpair_create`, 101
 - `gds_kvpair_destroy`, 102
 - KVPair, 101
- l
 - `gdt_generic_datatype`, 69
- length

- GDSSString, 68
- list, 72
- vector, 79
- List
 - Public interface to generic list data structure, 34
- list, 71
 - compfunc, 71
 - exit_on_error, 71
 - free_on_destroy, 72
 - head, 72
 - length, 72
 - list_string, 73
 - tail, 72
 - type, 72
- list.c
 - list_insert_internal, 140
 - list_node_at_index, 140
 - list_node_create, 140
 - list_node_destroy, 140
 - ListNode, 139
- list_append
 - Public interface to generic list data structure, 34
- list_create
 - Public interface to generic list data structure, 34
- list_delete_back
 - Public interface to generic list data structure, 34
- list_delete_front
 - Public interface to generic list data structure, 35
- list_delete_index
 - Public interface to generic list data structure, 35
- list_destroy
 - Public interface to generic list data structure, 35
- list_element_at_index
 - Public interface to generic list data structure, 35
- list_find
 - Public interface to generic list data structure, 36
- list_find_itr
 - Public interface to generic list data structure, 36
- list_get_value_itr
 - Public interface to generic list data structure, 36
- list_insert
 - Public interface to generic list data structure, 36
- list_insert_internal
 - list.c, 140
- list_is_empty
 - Public interface to generic list data structure, 37
- list_itr_first
 - Public interface to generic list data structure, 37
- list_itr_last
 - Public interface to generic list data structure, 37
- list_itr_next
 - Public interface to generic list data structure, 37
- list_itr_previous
 - Public interface to generic list data structure, 38
- list_length
 - Public interface to generic list data structure, 38
- list_node, 72
 - element, 73
 - next, 73
 - prev, 73
- list_node_at_index
 - list.c, 140
- list_node_create
 - list.c, 140
- list_node_destroy
 - list.c, 140
- list_prepend
 - Public interface to generic list data structure, 38
- list_reverse_sort
 - Public interface to generic list data structure, 38
- list_set_element_at_index
 - Public interface to generic list data structure, 39
- list_sort
 - Public interface to generic list data structure, 39
- list_string, 73
 - list, 73
 - size, 73
- list_string_create
 - General purpose string manipulation functions, 51
- list_string_destroy
 - General purpose string manipulation functions, 51
- list_string_resize
 - string_util.c, 145
- Listltr
 - Public interface to generic list data structure, 34
- ListNode
 - list.c, 139
- ll
 - gdt_generic_datatype, 69
- log_error
 - Public general generic data structures functionality, 29
- log_strerror
 - Public general generic data structures functionality, 29
- next
 - list_node, 73
- nonopts
 - gds_opt.c, 121
- num_buckets
 - dict, 66
- options
 - gds_opt.c, 121
- p
 - gdt_generic_datatype, 70
- pair_string, 74
 - first, 74
 - second, 74
- pair_string_copy
 - General purpose string manipulation functions, 51
- pair_string_create
 - General purpose string manipulation functions, 52
- pair_string_destroy
 - General purpose string manipulation functions, 52

- parsed
 - gds_opt.c, [121](#)
- pc
 - gdt_generic_datatype, [70](#)
- prev
 - list_node, [73](#)
- Private functionality for manipulating generic datatypes, [24](#)
 - DATATYPE_CHAR, [25](#)
 - DATATYPE_DOUBLE, [25](#)
 - DATATYPE_GDSSTRING, [25](#)
 - DATATYPE_INT, [25](#)
 - DATATYPE_LONG, [25](#)
 - DATATYPE_LONG_LONG, [25](#)
 - DATATYPE_POINTER, [25](#)
 - DATATYPE_SIGNED_CHAR, [25](#)
 - DATATYPE_SIZE_T, [25](#)
 - DATATYPE_STRING, [25](#)
 - DATATYPE_UNSIGNED_CHAR, [25](#)
 - DATATYPE_UNSIGNED_INT, [25](#)
 - DATATYPE_UNSIGNED_LONG, [25](#)
 - DATATYPE_UNSIGNED_LONG_LONG, [25](#)
 - gds_cfunc, [24](#)
 - gds_datatype, [25](#)
 - gdt_compare, [25](#)
 - gdt_compare_void, [25](#)
 - gdt_free, [26](#)
 - gdt_get_value, [26](#)
 - gdt_reverse_compare_void, [26](#)
 - gdt_set_value, [26](#)
- programe
 - gds_opt.c, [121](#)
- Public general generic data structures functionality, [28](#)
 - abort_error, [29](#)
 - GDS_EXIT_ON_ERROR, [31](#)
 - GDS_FREE_ON_DESTROY, [31](#)
 - GDS_RESIZABLE, [31](#)
 - gds_assert, [29](#)
 - gds_logerror_line, [31](#)
 - gds_option, [31](#)
 - gds_strdup, [32](#)
 - log_error, [29](#)
 - log_strerror, [29](#)
 - quit_error, [30](#)
 - quit_strerror, [30](#)
 - xcalloc, [30](#)
 - xfopen, [30](#)
 - xmalloc, [31](#)
 - xrealloc, [31](#)
 - xstrdup, [31](#)
- Public interface to command line parsing functionality, [11](#)
 - gds_free_options, [11](#)
 - gds_option_argument_int, [11](#)
 - gds_option_argument_string, [12](#)
 - gds_option_nonopty, [12](#)
 - gds_option_nonopty_number, [12](#)
 - gds_option_present, [12](#)
 - gds_option_programe, [13](#)
 - gds_parse_options, [13](#)
- Public interface to generic list data structure, [33](#)
 - List, [34](#)
 - list_append, [34](#)
 - list_create, [34](#)
 - list_delete_back, [34](#)
 - list_delete_front, [35](#)
 - list_delete_index, [35](#)
 - list_destroy, [35](#)
 - list_element_at_index, [35](#)
 - list_find, [36](#)
 - list_find_itr, [36](#)
 - list_get_value_itr, [36](#)
 - list_insert, [36](#)
 - list_is_empty, [37](#)
 - list_itr_first, [37](#)
 - list_itr_last, [37](#)
 - list_itr_next, [37](#)
 - list_itr_previous, [38](#)
 - list_length, [38](#)
 - list_prepend, [38](#)
 - list_reverse_sort, [38](#)
 - list_set_element_at_index, [39](#)
 - list_sort, [39](#)
 - ListItr, [34](#)
- Public interface to generic queue data structure, [41](#)
 - Queue, [41](#)
 - queue_capacity, [41](#)
 - queue_create, [42](#)
 - queue_destroy, [42](#)
 - queue_free_space, [42](#)
 - queue_is_empty, [42](#)
 - queue_is_full, [43](#)
 - queue_peek, [43](#)
 - queue_pop, [43](#)
 - queue_push, [43](#)
 - queue_size, [44](#)
- Public interface to generic stack data structure, [45](#)
 - Stack, [45](#)
 - stack_capacity, [45](#)
 - stack_create, [46](#)
 - stack_destroy, [46](#)
 - stack_free_space, [46](#)
 - stack_is_empty, [46](#)
 - stack_is_full, [47](#)
 - stack_peek, [47](#)
 - stack_pop, [47](#)
 - stack_push, [47](#)
 - stack_size, [48](#)
- Public interface to generic vector data structure., [59](#)
 - Vector, [60](#)
 - vector_append, [60](#)
 - vector_capacity, [60](#)
 - vector_create, [60](#)
 - vector_delete_back, [61](#)
 - vector_delete_front, [61](#)
 - vector_delete_index, [61](#)

- vector_destroy, 61
- vector_element_at_index, 62
- vector_find, 62
- vector_free_space, 62
- vector_insert, 62
- vector_is_empty, 63
- vector_length, 63
- vector_prepend, 63
- vector_reverse_sort, 64
- vector_set_element_at_index, 64
- vector_sort, 64
- Public interface to logging functionality, 40
 - gds_errlog, 40
 - gds_logging_off, 40
 - gds_logging_on, 40
- Public interface to string data structure, 14
 - GDSString, 15
 - GDSString_destructor, 23
 - gds_str_assign, 15
 - gds_str_assign_cstr, 15
 - gds_str_char_at_index, 16
 - gds_str_clear, 16
 - gds_str_compare, 16
 - gds_str_compare_cstr, 16
 - gds_str_concat, 17
 - gds_str_concat_cstr, 17
 - gds_str_create, 17
 - gds_str_create_direct, 17
 - gds_str_create_sprintf, 18
 - gds_str_cstr, 18
 - gds_str_decorate, 18
 - gds_str_destroy, 18
 - gds_str_doubleval, 19
 - gds_str_dup, 19
 - gds_str_getline, 19
 - gds_str_getline_assign, 19
 - gds_str_hash, 20
 - gds_str_intval, 20
 - gds_str_is_alnum, 20
 - gds_str_is_empty, 20
 - gds_str_length, 21
 - gds_str_size_to_fit, 21
 - gds_str_split, 21
 - gds_str_strchr, 21
 - gds_str_substr_left, 22
 - gds_str_substr_right, 22
 - gds_str_trim, 22
 - gds_str_trim_leading, 22
 - gds_str_trim_trailing, 22
 - gds_str_trunc, 23
- Public interface to unit testing functionality, 53
 - RUN_CASE, 54
 - TEST_ASSERT_EQUAL, 54
 - TEST_ASSERT_FALSE, 54
 - TEST_ASSERT_TRUE, 56
 - TEST_CASE, 56
 - TEST_SUITE, 56
 - tests_assert_almost_equal, 56
 - tests_assert_true, 57
 - tests_get_failures, 57
 - tests_get_successes, 57
 - tests_get_total_tests, 57
 - tests_initialize, 57
 - tests_report, 58
- Queue
 - Public interface to generic queue data structure, 41
- queue, 75
 - back, 75
 - capacity, 75
 - elements, 75
 - exit_on_error, 76
 - free_on_destroy, 76
 - front, 76
 - resizable, 76
 - size, 76
 - type, 76
- queue.c
 - GROWTH, 142
- queue_capacity
 - Public interface to generic queue data structure, 41
- queue_create
 - Public interface to generic queue data structure, 42
- queue_destroy
 - Public interface to generic queue data structure, 42
- queue_free_space
 - Public interface to generic queue data structure, 42
- queue_is_empty
 - Public interface to generic queue data structure, 42
- queue_is_full
 - Public interface to generic queue data structure, 43
- queue_peek
 - Public interface to generic queue data structure, 43
- queue_pop
 - Public interface to generic queue data structure, 43
- queue_push
 - Public interface to generic queue data structure, 43
- queue_size
 - Public interface to generic queue data structure, 44
- quit_error
 - Public general generic data structures functionality, 30
- quit_strerror
 - Public general generic data structures functionality, 30
- RUN_CASE
 - Public interface to unit testing functionality, 54
- resizable
 - queue, 76
 - stack, 78
- sc
 - gdt_generic_datatype, 70
- second
 - pair_string, 74
- show_failures

- test_logging.c, 146
- size
 - list_string, 73
 - queue, 76
- split_string
 - General purpose string manipulation functions, 52
- src/dict.c, 114
- src/gds_opt.c, 118
- src/gds_string.c, 121
- src/gds_util_error.c, 126
- src/gds_util_logging.c, 127
- src/gds_util_std_wrappers.c, 128
- src/gdt.c, 131
- src/kvpair.c, 136
- src/list.c, 138
- src/queue.c, 141
- src/stack.c, 142
- src/string_util.c, 144
- src/test_logging.c, 145
- src/vector.c, 147
- st
 - gdt_generic_datatype, 70
- Stack
 - Public interface to generic stack data structure, 45
- stack, 77
 - capacity, 77
 - elements, 77
 - exit_on_error, 77
 - free_on_destroy, 77
 - resizable, 78
 - top, 78
 - type, 78
- stack.c
 - GROWTH, 143
- stack_capacity
 - Public interface to generic stack data structure, 45
- stack_create
 - Public interface to generic stack data structure, 46
- stack_destroy
 - Public interface to generic stack data structure, 46
- stack_free_space
 - Public interface to generic stack data structure, 46
- stack_is_empty
 - Public interface to generic stack data structure, 46
- stack_is_full
 - Public interface to generic stack data structure, 47
- stack_peek
 - Public interface to generic stack data structure, 47
- stack_pop
 - Public interface to generic stack data structure, 47
- stack_push
 - Public interface to generic stack data structure, 47
- stack_size
 - Public interface to generic stack data structure, 48
- string_util.c
 - list_string_resize, 145
- TEST_ASSERT_EQUAL
 - Public interface to unit testing functionality, 54
- TEST_ASSERT_FALSE
 - Public interface to unit testing functionality, 54
- TEST_ASSERT_TRUE
 - Public interface to unit testing functionality, 56
- TEST_CASE
 - Public interface to unit testing functionality, 56
- TEST_SUITE
 - Public interface to unit testing functionality, 56
- tail
 - list, 72
- test_failures
 - test_logging.c, 147
- test_logging.c
 - show_failures, 146
 - test_failures, 147
 - test_successes, 147
 - tests_log_single_test, 146
 - total_tests, 147
- test_successes
 - test_logging.c, 147
- tests_assert_almost_equal
 - Public interface to unit testing functionality, 56
- tests_assert_true
 - Public interface to unit testing functionality, 57
- tests_get_failures
 - Public interface to unit testing functionality, 57
- tests_get_successes
 - Public interface to unit testing functionality, 57
- tests_get_total_tests
 - Public interface to unit testing functionality, 57
- tests_initialize
 - Public interface to unit testing functionality, 57
- tests_log_single_test
 - test_logging.c, 146
- tests_report
 - Public interface to unit testing functionality, 58
- top
 - stack, 78
- total_tests
 - test_logging.c, 147
- truncate_if_needed
 - gds_string.c, 126
- type
 - dict, 66
 - gdt_generic_datatype, 70
 - list, 72
 - queue, 76
 - stack, 78
 - vector, 79
- uc
 - gdt_generic_datatype, 70
- ui
 - gdt_generic_datatype, 70
- ul
 - gdt_generic_datatype, 70
- ull
 - gdt_generic_datatype, 70

- value
 - gds_kvpair, [67](#)
- Vector
 - Public interface to generic vector data structure., [60](#)
- vector, [78](#)
 - capacity, [79](#)
 - compfunc, [79](#)
 - elements, [79](#)
 - exit_on_error, [79](#)
 - free_on_destroy, [79](#)
 - length, [79](#)
 - type, [79](#)
- vector.c
 - GROWTH, [149](#)
 - vector_insert_internal, [149](#)
- vector_append
 - Public interface to generic vector data structure., [60](#)
- vector_capacity
 - Public interface to generic vector data structure., [60](#)
- vector_create
 - Public interface to generic vector data structure., [60](#)
- vector_delete_back
 - Public interface to generic vector data structure., [61](#)
- vector_delete_front
 - Public interface to generic vector data structure., [61](#)
- vector_delete_index
 - Public interface to generic vector data structure., [61](#)
- vector_destroy
 - Public interface to generic vector data structure., [61](#)
- vector_element_at_index
 - Public interface to generic vector data structure., [62](#)
- vector_find
 - Public interface to generic vector data structure., [62](#)
- vector_free_space
 - Public interface to generic vector data structure., [62](#)
- vector_insert
 - Public interface to generic vector data structure., [62](#)
- vector_insert_internal
 - vector.c, [149](#)
- vector_is_empty
 - Public interface to generic vector data structure., [63](#)
- vector_length
 - Public interface to generic vector data structure., [63](#)
- vector_prepend
 - Public interface to generic vector data structure., [63](#)
- vector_reverse_sort
 - Public interface to generic vector data structure., [64](#)
- vector_set_element_at_index
 - Public interface to generic vector data structure., [64](#)
- vector_sort
 - Public interface to generic vector data structure., [64](#)
- xalloc
 - Public general generic data structures functionality, [30](#)
- xfopen
 - Public general generic data structures functionality, [30](#)
- xmalloc
 - Public general generic data structures functionality, [31](#)
- xrealloc
 - Public general generic data structures functionality, [31](#)
- xstrdup
 - Public general generic data structures functionality, [31](#)