gds

Generated by Doxygen 1.8.1.2

Sun Nov 30 2014 19:54:04

Contents

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:

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

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_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.

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

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

Return values

non-NULL	The string argument
NULL	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

optname	A string containing the single-character option name.

Return values

non-NULL	The string argument
NULL	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

index	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

optname	A string containing the single-character option name.

Return values

true	Option was provided
false	Option was not provided, or command line has not yet been parsed.

6.1.2.7 const char* gds_option_progname (void)

Returns the program name.

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

Return values

non-NULL	The program name
NULL	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

allowed	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.
argv	List of command line strings passed to `main()`.

Return values

true	Success
false	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 GDSString ∗ GDSString

  Opaque data type for string.

Functions

- 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.
- GDSString gds_str_create_direct (char ∗init_str, const size_t init_str_size)

  Creates a string using allocated memory.
- void gds_str_destroy (GDSString str)

  Destroys a string and releases allocated resources.
- void GDSString_destructor (void ∗str)

  Destroys a string and releases allocated resources.
- GDSString gds_str_assign (GDSString dst, GDSString src)

  Assigns a string to another.
- GDSString gds_str_assign_cstr (GDSString dst, const char ∗src)

  Assigns a C-style string to a string.
- const char ∗ gds_str_cstr (GDSString str)

  Returns a C-style string containing the string's contents.
- size_t gds_str_length (GDSString str)

  Returns the length of a string.
- GDSString gds_str_size_to_fit (GDSString str)

  Reduces a string's capacity to fit its length.
- GDSString gds_str_concat (GDSString dst, GDSString src)

  Concatenates two strings.
- GDSString gds_str_concat_cstr (GDSString dst, const char ∗src)

  Concatenates a C-style string to a string.
- GDSString gds_str_trunc (GDSString str, const size_t length)

  Truncates a string.
- unsigned long gds_str_hash (GDSString str)

  Calculates a hash of a string.
- int gds_str_compare (GDSString s1, GDSString s2)

  Compares two strings.
- int gds_str_compare_cstr (GDSString s1, const char ∗s2)

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

  Returns index of first occurence of a character.
- GDSString gds_str_substr_left (GDSString str, const size_t numchars)

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

  Returns a right substring.
- void gds_str_split (GDSString src, GDSString ∗left, GDSString ∗right, const char sc)

Splits a string.
- void gds_str_trim_leading (GDSString str)

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

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

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

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

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

  Checks is a string contains only alphanumeric characters.
- void gds_str_clear (GDSString str)

  Clears (empties) a string.
- bool gds_str_intval (GDSString str, const int base, int ∗value)

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

  Gets the double value of a string.
- GDSString gds_str_getline (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.

6.2.1 Detailed Description

A string is an ordered collection of characters.

6.2.2 Typedef Documentation

6.2.2.1 typedef struct GDSString∗ GDSString

Opaque data type for string.

6.2.3 Function Documentation

6.2.3.1 GDSString gds_str_assign (GDSString *dst*, GDSString *src*)

Assigns a string to another.

Parameters

dst	The destination string.
src	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

dst	The destination string.
src	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

str	The string.
index	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

str	The string.

6.2.3.5 int gds_str_compare (GDSString *s1*, GDSString *s2*)

Compares two strings.

Parameters

s1	The first string.
s2	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

s1	The first string.
s2	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

dst	The destination string.
src	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

dst	The destination string.
src	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

init_str	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

init_str	The allocated memory. IMPORTANT: If the construction of the string fails, this memory will be `free()`d.
init_str_size	The size of the allocated memory. IMPORTANT: The string's length is assumed to be one less than this quantity, and a call to `strlen()` 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

format	The format string.
...	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

str	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

str	The string to decorate.
left_dec	The string to add to the left of `str`.
right_dec	The string to add to the right of `str`, or `NULL` to add `left_dec` 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

str	The string to destroy..

6.2.3.15 bool gds_str_doubleval (GDSString *str*, double * *value*)

Gets the double value of a string.

Parameters

str	The string.
value	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 GDSString gds_str_dup (GDSString *src*)

Creates a new string from another string.

Parameters

src	The other string.

Returns

The new string, or `NULL` on failure.

6.2.3.17 GDSString 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

size	The maximum number of bytes to read, including the null.
fp	The file pointer from which to read.

Returns

`dst`

6.2.3.18 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.

Any trailing newline character is stripped.

Parameters

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

Returns

`dst`

6.2.3.19 unsigned long gds_str_hash (GDSString *str*)

Calculates a hash of a string.

Uses Dan Bernstein's djb2 algorithm.

Parameters

str	The string.

Returns

The hash value

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

Gets the integer value of a string.

Parameters

str	The string.
base	The base of the integer. This has the same meaning as the third argument to standard C `strtol()`.
value	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 (GDSString *str*)

Checks is 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

str	The string.

Returns

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

6.2.3.22 bool gds_str_is_empty (GDSString *str*)

Checks if a string is empty.

Parameters

str	The string.

Returns

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

6.2.3.23 size_t gds_str_length (**GDSString** *str*)

Returns the length of a string.

Parameters

str	The string.

Returns

The length of the string.

6.2.3.24 **GDSString** gds_str_size_to_fit (**GDSString** *str*)

Reduces a string's capacity to fit its length.

Parameters

str	The string to size.

Returns

`str`, or `NULL` on failure.

6.2.3.25 void gds_str_split (**GDSString** *src,* **GDSString** $*$ *left,* **GDSString** $*$ *right,* const char *sc*)

Splits a string.

Parameters

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

6.2.3.26 int gds_str_strchr (**GDSString** *str,* const char *ch,* const int *start*)

Returns index of first occurence of a character.

Parameters

str	The string.
ch	The character for which to search.
start	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 occurence, 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

str	The string.
numchars	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

str	The string.
numchars	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

str	The string.

6.2.3.30 void gds_str_trim_leading (GDSString *str*)

Trims leading whitespace in-place.

Parameters

str	The string.

6.2.3.31 void gds_str_trim_trailing (GDSString *str*)

Trims trailing whitespace in-place.

Parameters

str	The string.

6.2.3.32 GDSString gds_str_trunc (GDSString *str*, const size_t *length*)

Truncates a string.

Parameters

str	The string.
length	The new length to which to truncate.

Returns

The original string, or `NULL` on failure.

6.2.3.33 void GDSString_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

str	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_STRIN-G,
  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

d1	A pointer to the first generic datatype.
d2	A pointer to the second generic datatype.

Return values

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

6.3.4.2 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

p1	A pointer to the first generic datatype.
p2	A pointer to the second generic datatype.

Return values

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

6.3.4.3 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

data	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

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

6.3.4.5 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

p1	A pointer to the first generic datatype.
p2	A pointer to the second generic datatype.

Return values

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

6.3.4.6 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

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

6.4 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 ∗progname, 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

prog	The program name to include in the error message.
...	Other arguments, the first of which should be a format string suitable for passing to `vprintf()`, 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

cond	The assertion to test.
prog	The program name to include in the error message.
...	Other arguments, the first of which should be a format string suitable for passing to `vprintf()`, 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

prog	The program name to include in the error message.
...	Other arguments, the first of which should be a format string suitable for passing to `vprintf()`, 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

prog	The program name to include in the error message.
...	Other arguments, the first of which should be a format string suitable for passing to `vprintf()`, 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

prog	The program name to include in the error message.
...	Other arguments, the first of which should be a format string suitable for passing to `vprintf()`, 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

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

6.4.2.7 #define xcalloc(*n, s*) gds_xcalloc((n), (s), __FILE__, __LINE__)

Macro to call calloc() and abort on failure.

Parameters

n	The number of members to allocate.
s	The size in bytes of each member.

6.4.2.8 #define xfopen(*path, mode*) gds_xfopen((path), (mode), __FILE__, __LINE__)

Macro to call strdup() and abort on failure.

Parameters

path	The path of the file to open.
mode	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

s	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

p	A pointer to the memory to reallocate.
s	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

str	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 * *progname,* 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.

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

Parameters

progname	The program name to include in the message.
filename	The name of the source file.
linenum	The line number of the source file.
log_errno	Set to `true` to include the current value of `errno` and the string representation of that error in the message.
quit_type	Info on how to quit the function.
fmt	The format string for the message to print. Format specifiers are the same as the `printf()` family of functions.
...	Any arguments to the format string.

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

Dynamically duplicates a string.

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

Parameters

str	The string to duplicate.

Return values

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

6.5 Public interface to generic list data structure

Typedefs

- typedef struct list ∗ List
  Opaque list type definition.
- typedef struct list_node ∗ ListItr
  Opaque list iterator type definition.

Functions

- List list_create (const enum gds_datatype type, const int opts,...)
  Creates a new list.
- void list_destroy (List list)
  Destroys a list.
- bool list_append (List list,...)
  Appends a value to the back of a list.
- bool list_prepend (List list,...)
  Prepends a value to the front of a list.
- bool list_insert (List list, const size_t index,...)
  Inserts a value into a list.
- bool list_delete_front (List list)
  Deletes the value at the front of the list.
- bool list_delete_back (List list)
  Deletes the value at the back of the list.
- bool list_delete_index (List list, const size_t index)
  Deletes the value at the specified index of the list.
- bool list_element_at_index (List list, const size_t index, void ∗p)
  Gets the value at the specified index of the list.
- bool list_set_element_at_index (List list, const size_t index,...)
  Sets the value at the specified index of the list.
- bool list_find (List list, size_t ∗index,...)
  Tests if a value is contained in a list.
- ListItr list_find_itr (List list,...)
  Tests if a value is contained in a list.
- bool list_sort (List list)
  Sorts a list in-place, in ascending order.
- bool list_reverse_sort (List list)
  Sorts a list in-place, in descending order.
- ListItr list_itr_first (List list)
  Returns an iterator to the first element of the list.
- ListItr list_itr_last (List list)
  Returns an iterator to the last element of the list.
- ListItr list_itr_next (ListItr itr)
  Increments a list iterator.
- ListItr list_itr_previous (ListItr itr)
  Decrements a list iterator.
- void list_get_value_itr (ListItr itr, void ∗p)
  Retrieves a value from an iterator.
- bool list_is_empty (List list)
  Tests if a list is empty.
- size_t list_length (List list)
  Returns the length of a list.

6.5.1 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

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

Return values

true	Success
false	Failure, dynamic memory allocation failed.

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

Creates a new list.

Parameters

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

Return values

NULL	List creation failed.
non-NULL	A pointer to the new list.

6.5.3.3 bool list_delete_back (List *list*)

Deletes the value at the back of the list.

Parameters

list	A pointer to the list.

Return values

true	Success
false	Failure, dynamic memory allocation failed.

6.5.3.4 bool list_delete_front (List *list*)

Deletes the value at the front of the list.

Parameters

list	A pointer to the list.

Return values

true	Success
false	Failure, dynamic memory allocation failed.

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

Deletes the value at the specified index of the list.

Parameters

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

Return values

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

6.5.3.6 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

list	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

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

Return values

true	Success
false	Failure, index was out of range.

6.5.3.8 bool list_find (List *list*, size_t ∗ *index*, ...)

Tests if a value is contained in a list.

Parameters

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

Return values

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

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

Tests if a value is contained in a list.

Parameters

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

Return values

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

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

Retrieves a value from an iterator.

Parameters

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

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

Inserts a value into a list.

Parameters

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

Return values

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

6.5.3.12 bool list_is_empty (List *list*)

Tests if a list is empty.

Parameters

list	A pointer to the list.

Return values

true	The list is empty
false	The list is not empty

6.5.3.13 ListItr list_itr_first (List *list*)

Returns an iterator to the first element of the list.

Parameters

list	A pointer to the list

Return values

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

6.5.3.14 ListItr list_itr_last (List *list*)

Returns an iterator to the last element of the list.

Parameters

list	A pointer to the list

Return values

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

6.5.3.15 ListItr list_itr_next (ListItr *itr*)

Increments a list iterator.

Parameters

itr	A pointer to the iterator.

Return values

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

6.5.3.16 ListItr list_itr_previous (ListItr *itr*)

Decrements a list iterator.

Parameters

itr	A pointer to the iterator.

Return values

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

6.5.3.17 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

list	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

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

Return values

true	Success
false	Failure, dynamic memory allocation failed.

6.5.3.19 bool list_reverse_sort (List *list*)

Sorts a list in-place, in descending order.

Parameters

list	A pointer to the list.

Return values

true	Success
false	Failure, dynamic memory allocation failed.

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

Sets the value at the specified index of the list.

Parameters

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

Return values

true	Success
false	Failure, index was out of range.

6.5.3.21 bool list_sort (List *list*)

Sorts a list in-place, in ascending order.

Parameters

list	A pointer to the list.

Return values

true	Success
false	Failure, dynamic memory allocation failed.

6.6 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

true	Success
false	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

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

Return values

true	Success
false	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

queue	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

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

Return values

NULL	Queue creation failed.
non-NULL	A pointer to the new queue.

6.7.3.3 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

queue	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

queue	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

queue	A pointer to the queue.

Return values

true	Queue is empty
false	Queue is not empty

6.7.3.6 bool queue_is_full (Queue *queue*)

Checks whether a queue is full.

Parameters

queue	A pointer to the queue.

Return values

true	Queue is full
false	Queue is not full

6.7.3.7 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

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

Return values

true	Success
false	Failure, queue is empty.

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

Pops a value from the queue.

Parameters

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

Return values

true	Success
false	Failure, queue is empty.

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

Pushes a value onto the queue.

Parameters

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

Return values

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

Todo Rewrite to move only the required elements

6.7.3.10 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

queue	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

stack	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

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

Return values

NULL	Stack creation failed.
non-NULL	A pointer to the new stack.

6.8.3.3 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

stack	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

stack	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

stack	A pointer to the stack.

Return values

true	Stack is empty
false	Stack is not empty

6.8.3.6 bool stack_is_full (Stack *stack*)

Checks whether a stack is full.

Parameters

stack	A pointer to the stack.

Return values

true	Stack is full
false	Stack is not full

6.8.3.7 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

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

Return values

true	Success
false	Failure, stack is empty.

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

Pops a value from the stack.

Parameters

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

Return values

true	Success
false	Failure, stack is empty.

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

Pushes a value onto the stack.

Parameters

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

Return values

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

6.8.3.10 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

stack	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

str	The string to duplicate.

Return values

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

Duplicates a string.

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

Parameters

str	The string to duplicate.

Return values

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

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

Duplicates at most n characters of a string.

Parameters

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

Return values

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

6.9.2.3 char∗ gds_trim (char ∗ *str*)

Trims leading and trailing whitespace from a string.

Parameters

str	The string to trim.

Returns

A pointer to the passed string.

6.9.2.4 char∗ gds_trim_left (char ∗ *str*)

Trims leading whitespace from a string.

Parameters

str	The string to trim.

Returns

A pointer to the passed string.

6.9.2.5 char* gds_trim_line_ending (char * *str*)

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

Parameters

str	The string to trim.

Returns

A pointer to the passed string.

6.9.2.6 char* gds_trim_right (char * *str*)

Trims trailing whitespace from a string.

Parameters

str	The string to trim.

Returns

A pointer to the passed string.

6.9.2.7 struct list_string* list_string_create (const size_t *n*) `[read]`

Creates a string list.

Parameters

n	The capacity of the string list.

Return values

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

6.9.2.8 void list_string_destroy (struct list_string * *list*)

Destroys a string list.

Parameters

list	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

pair	The string pair to copy.

Return values

NULL	Failure, dynamic memory allocation failed
non-NULL	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

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

Return values

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

6.9.2.11 void pair_string_destroy (struct pair_string ∗ *pair*)

Destroys a string pair.

Parameters

pair	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

str	The string to split.
delim	The delimiter character.

Return values

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

6.10 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

name	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

a	The first number.
b	The second number.
e	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

a	The first value.
b	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

cond	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

a	The first value.
b	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

s1	The first string.
s2	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

s1	The first string.
s2	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

cond	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

name	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

name	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

a	The first number.
b	The second number.
e	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

true	The numbers are equal to the specified precision
false	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

success	The test condition.
suitename	The name of the test suite.
casename	The name of the test case.
failmessage	The message to print on test failure.
filename	The name of the file containing the test.
linenum	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

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

Return values

true	Success
false	Failure, dynamic memory allocation failed.

6.11.3.2 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

vector	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

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

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

Parameters

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

Return values

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

6.11.3.4 bool vector_delete_back (Vector *vector*)

Deletes the value at the back of the vector.

Parameters

vector	A pointer to the vector.

Return values

true	Success
false	Failure, dynamic memory allocation failed.

6.11.3.5 bool vector_delete_front (Vector *vector*)

Deletes the value at the front of the vector.

Parameters

vector	A pointer to the vector.

Return values

true	Success
false	Failure, dynamic memory allocation failed.

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

Deletes the value at the specified index of the vector.

Parameters

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

Return values

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

6.11.3.7 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

vector	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

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

Return values

true	Success
false	Failure, index was out of range.

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

Tests if a value is contained in a vector.

Parameters

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

Return values

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

6.11.3.10 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

vector	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

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

Return values

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

6.11.3.12 bool vector_is_empty (Vector *vector*)

Tests if a vector is empty.

Parameters

vector	A pointer to the vector.

Return values

true	The vector is empty
false	The vector is not empty

6.11.3.13 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

vector	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

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

Return values

true	Success
false	Failure, dynamic memory allocation failed.

6.11.3.15 void vector_reverse_sort (Vector *vector*)

Sorts a vector in-place, in descending order.

Parameters

vector	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

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

Return values

true	Success
false	Failure, index was out of range.

6.11.3.17 void vector_sort (Vector *vector*)

Sorts a vector in-place, in ascending order.

Parameters

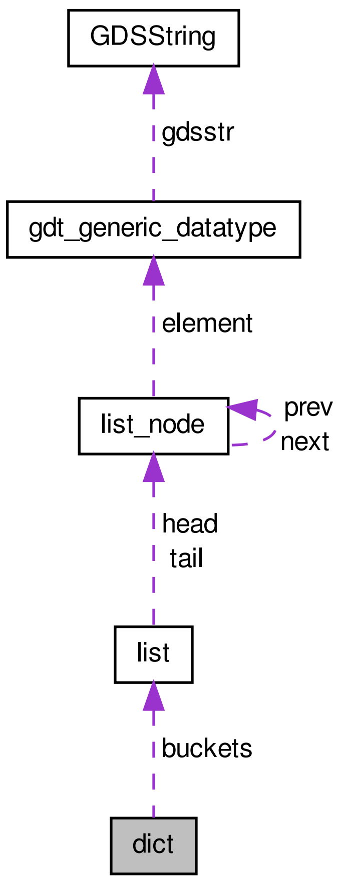
vector	A pointer to the vector.

Chapter 7

Data Structure Documentation

7.1 dict Struct Reference

Collaboration diagram for dict:



Data Fields

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

7.1.1 Detailed Description

Dict structure

7.1.2 Field Documentation

7.1.2.1 List∗ dict::buckets

The buckets

7.1.2.2 bool dict::exit_on_error

Exit on error if true

7.1.2.3 bool dict::free_on_destroy

Free pointer elements on destroy if true

7.1.2.4 size_t dict::num_buckets

Number of buckets

7.1.2.5 enum gds_datatype dict::type

Dict datatype

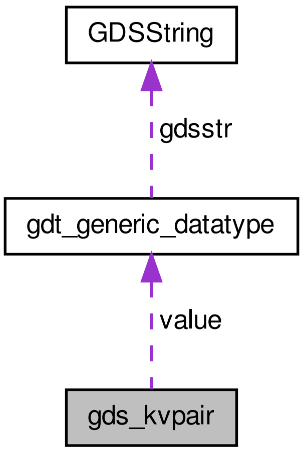
The documentation for this struct was generated from the following file:

- src/dict.c

7.2 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 GDSString 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 GDSString::capacity

The size of the `data` buffer

7.3.2.2 char* GDSString::data

The data in C-style string format

7.3.2.3 size_t GDSString::length

The length of the string

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

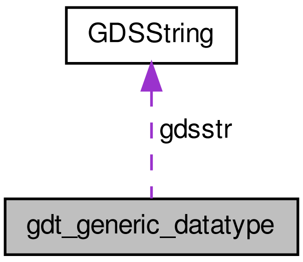
- src/gds_string.c

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
  GDSString 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 GDSString gdt_generic_datatype::gdsstr

GDSString

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

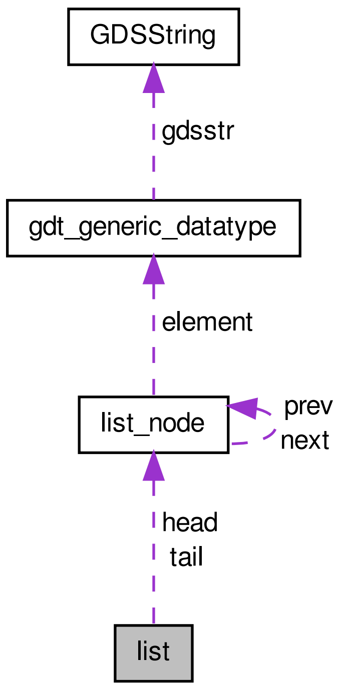
unsigned long long

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

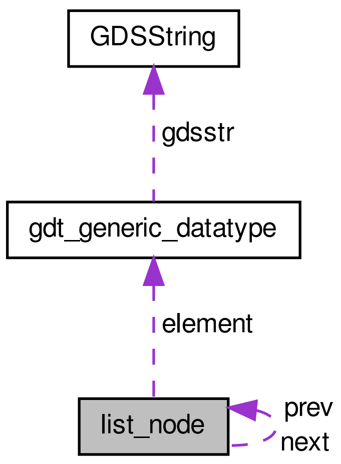
List datatype

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

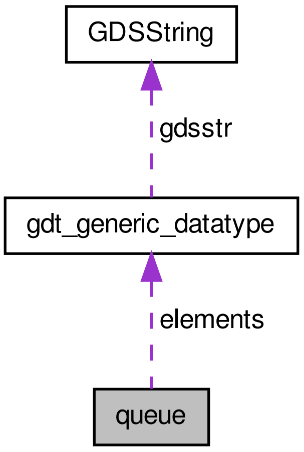
Second string of pair

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

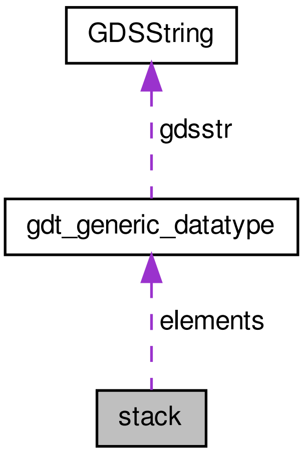
Queue datatype

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 resizabe if true

7.10.2.6 size_t stack::top

Top of stack

7.10.2.7 enum gds_datatype stack::type

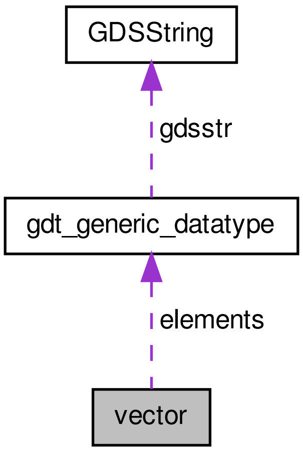
Stack datatype

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

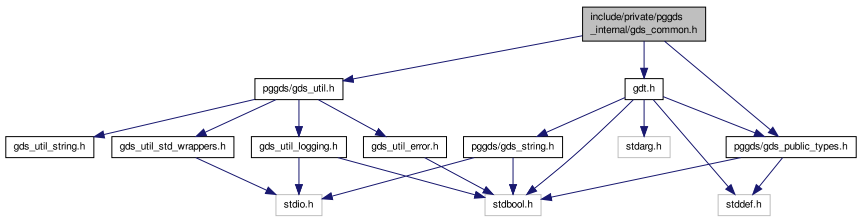
8.12 docs/vector.dox File Reference

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

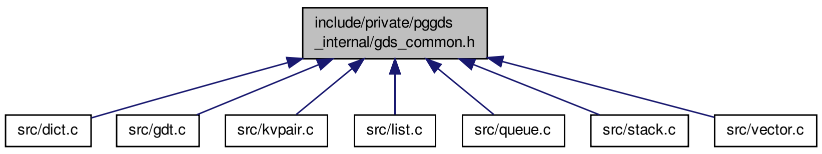
Common internal headers for data structures.

```
#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

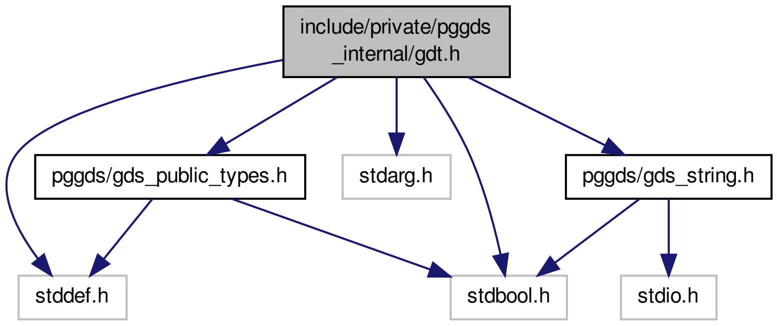
Copyright

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

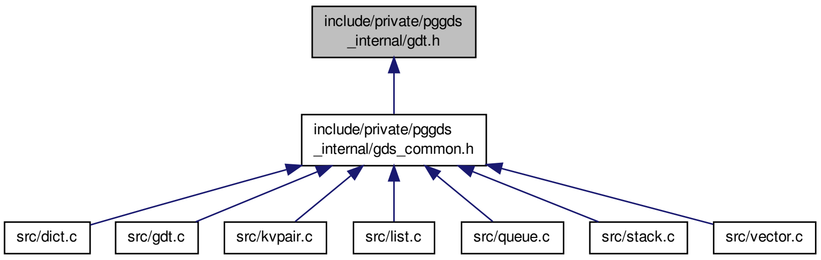
Interface to generic data element functionality.

```
#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:


include/private/pggds_internal/gdt.h
include/private/pggds_internal/gds_common.h
src/dict.c
src/gdt.c
src/kvpair.c
src/list.c
src/queue.c
src/stack.c
src/vector.c

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

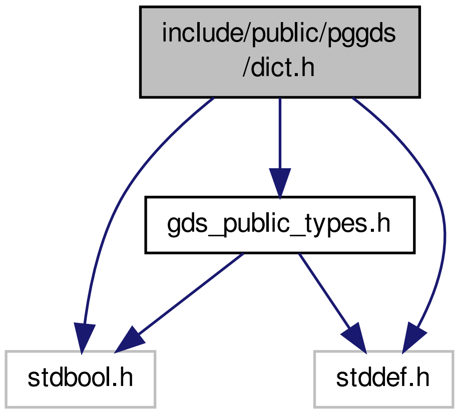
Copyright

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

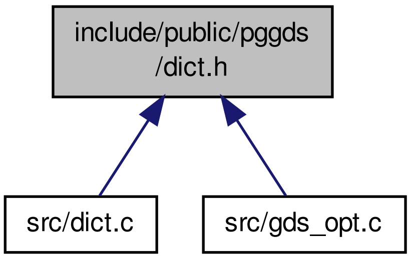
Interface to generic dictionary data structure.

```
#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

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

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

Return values

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

8.15.3.2 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

dict	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

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

Return values

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

8.15.3.4 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

dict	A pointer to the dictionary.
key	The key.

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

Return values

true	Success
false	Failure, dynamic memory allocation failed

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

Retrieves the value for a key in the dictionary.

Parameters

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

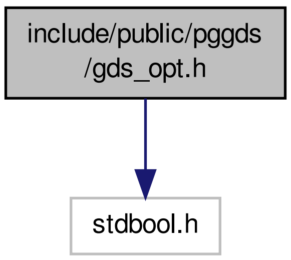
Return values

true	Success
false	Failure, key was not found

8.16 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

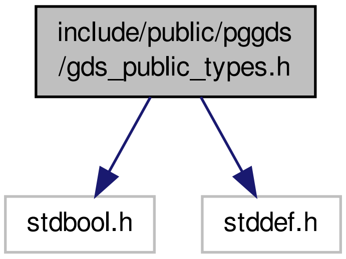
Copyright 2014 Paul Griffiths. Distributed under the terms of the GNU General Public License. `http://www.gnu.org/licenses/`

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

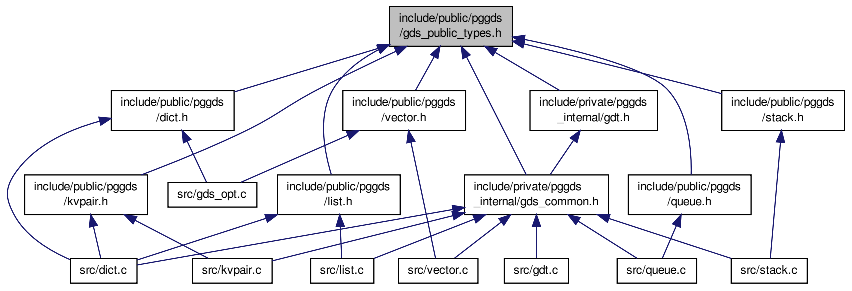
Common public types for generic data structures library.

```
#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:



include/public/pggds/gds_public_types.h
include/public/pggds/dict.h
include/public/pggds/vector.h
include/private/pggds_internal/gdt.h
include/public/pggds/stack.h
include/public/pggds/kvpair.h
src/gds_opt.c
include/public/pggds/list.h
include/private/pggds_internal/gds_common.h
include/public/pggds/queue.h
src/dict.c
src/kvpair.c
src/list.c
src/vector.c
src/gdt.c
src/queue.c
src/stack.c

Typedefs

- typedef int(∗ gds_cfunc)(const void ∗, const void ∗)

  Type definition for comparison function pointer.

Enumerations

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

  Enumeration type for data structure options.
- enum gds_datatype {
  DATATYPE_CHAR, DATATYPE_UNSIGNED_CHAR, DATATYPE_SIGNED_CHAR, DATATYPE_INT, DATATYPE_UNSIGNED_INT, DATATYPE_LONG, DATATYPE_UNSIGNED_LONG, DATATYPE_LONG_-LONG,
  DATATYPE_UNSIGNED_LONG_LONG, DATATYPE_SIZE_T, DATATYPE_DOUBLE, DATATYPE_STRIN-G,
  DATATYPE_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

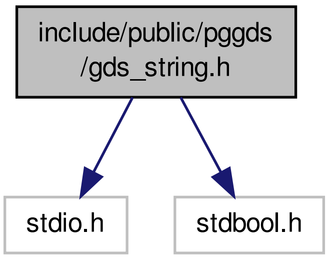
Copyright 2014 Paul Griffiths. Distributed under the terms of the GNU General Public License. http-://www.gnu.org/licenses/

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

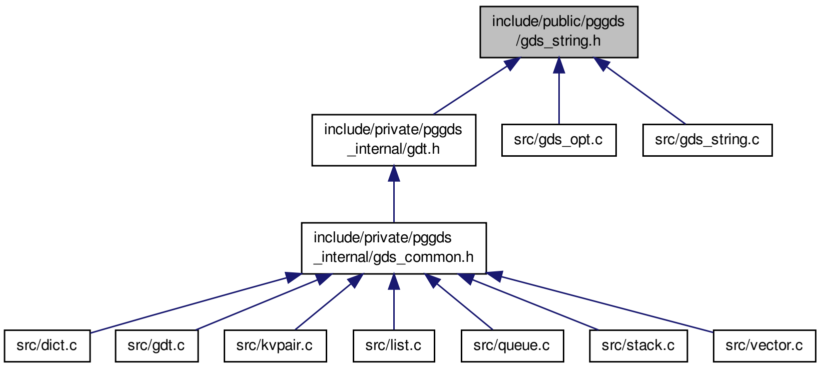
Interface to string data structure.

```
#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 GDSString ∗ GDSString

    Opaque data type for string.

Functions

- 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.

- GDSString gds_str_create_direct (char ∗init_str, const size_t init_str_size)

    Creates a string using allocated memory.

- void gds_str_destroy (GDSString str)

  Destroys a string and releases allocated resources.
- void GDSString_destructor (void ∗str)

  Destroys a string and releases allocated resources.
- GDSString gds_str_assign (GDSString dst, GDSString src)

  Assigns a string to another.
- GDSString gds_str_assign_cstr (GDSString dst, const char ∗src)

  Assigns a C-style string to a string.
- const char ∗ gds_str_cstr (GDSString str)

  Returns a C-style string containing the string's contents.
- size_t gds_str_length (GDSString str)

  Returns the length of a string.
- GDSString gds_str_size_to_fit (GDSString str)

  Reduces a string's capacity to fit its length.
- GDSString gds_str_concat (GDSString dst, GDSString src)

  Concatenates two strings.
- GDSString gds_str_concat_cstr (GDSString dst, const char ∗src)

  Concatenates a C-style string to a string.
- GDSString gds_str_trunc (GDSString str, const size_t length)

  Truncates a string.
- unsigned long gds_str_hash (GDSString str)

  Calculates a hash of a string.
- int gds_str_compare (GDSString s1, GDSString s2)

  Compares two strings.
- int gds_str_compare_cstr (GDSString s1, const char ∗s2)

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

  Returns index of first occurence of a character.
- GDSString gds_str_substr_left (GDSString str, const size_t numchars)

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

  Returns a right substring.
- void gds_str_split (GDSString src, GDSString ∗left, GDSString ∗right, const char sc)

  Splits a string.
- void gds_str_trim_leading (GDSString str)

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

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

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

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

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

  Checks is a string contains only alphanumeric characters.
- void gds_str_clear (GDSString str)

  Clears (empties) a string.
- bool gds_str_intval (GDSString str, const int base, int ∗value)

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

Gets the double value of a string.
- GDSString gds_str_getline (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

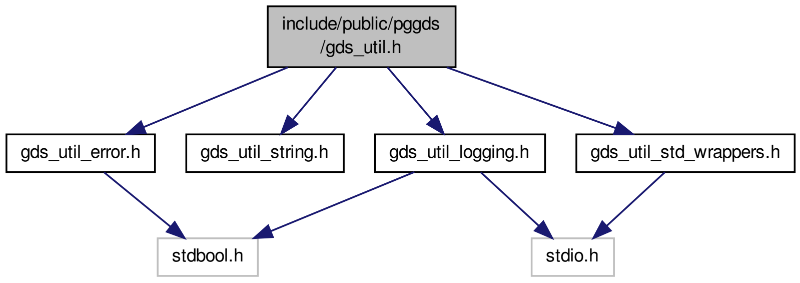
Copyright

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

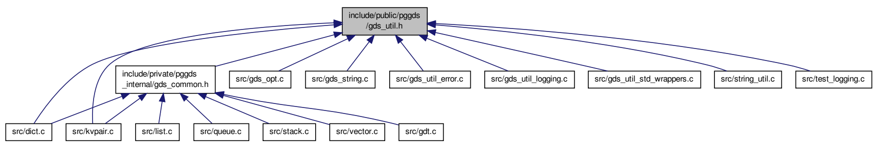
8.19 include/public/pggds/gds_util.h File Reference

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

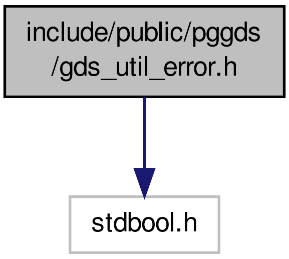
Paul Griffiths

Copyright

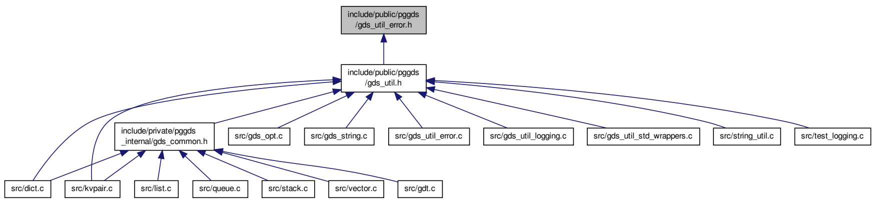
8.20 include/public/pggds/gds_util_error.h File Reference

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 ∗progname, 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

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

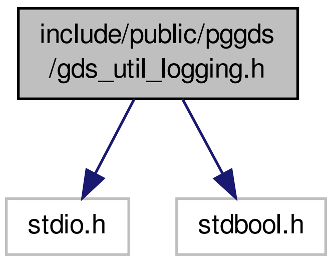
GDS_ERROR_ASSERT

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

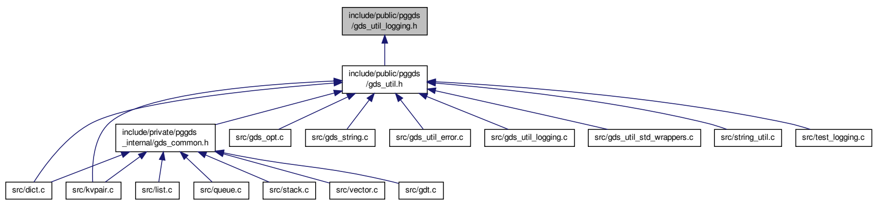
Interface to logging functions.

```
#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

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

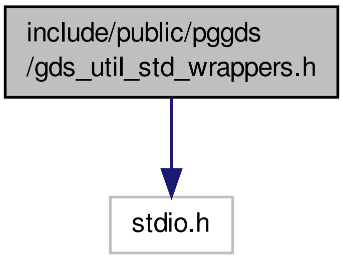
8.21.3.1 void gds_log_msg (const char ∗ *fmt*, ...)

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

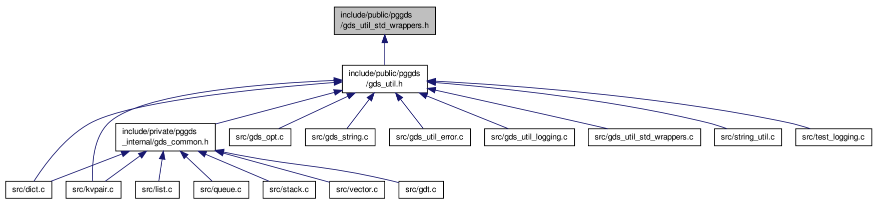
Interface to wrappers for standard functions.

`#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

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

nmemb	The number of members to allocate.
size	The size in bytes of each member.
filename	The name of the calling file.
linenum	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

path	The path of the file to open.
mode	The mode under which to open the file.
filename	The name of the calling file.
linenum	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

size	The number of bytes to allocate.
filename	The name of the calling file.
linenum	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

ptr	A pointer to the memory to reallocate.
size	The number of bytes for the new allocation.
filename	The name of the calling file.
linenum	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

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

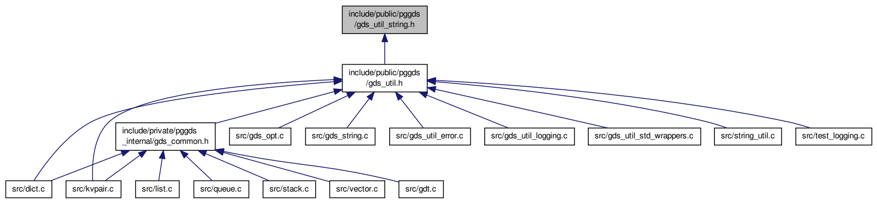
Returns

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

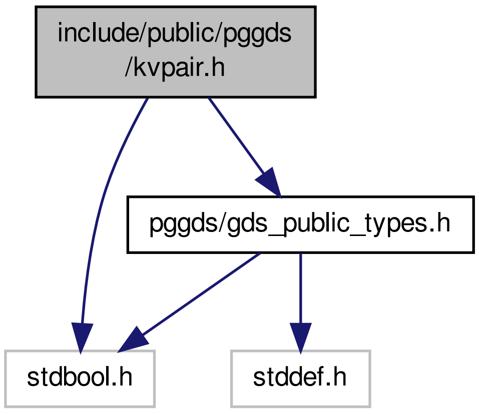
Copyright 2014 Paul Griffiths. Distributed under the terms of the GNU General Public License. http://www.gnu.org/licenses/

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

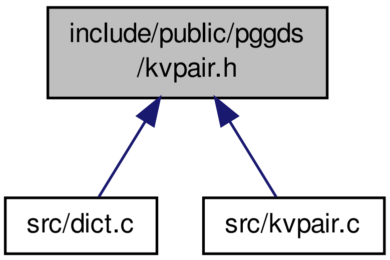
Interface to generic key-value pair structure.

```
#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

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

Return values

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

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

Creates a new key-value pair.

Parameters

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

Return values

NULL	Failure, dynamic memory allocation failed
non-NULL	Success

8.24.3.3 void gds_kvpair_destroy (KVPair *pair*, const bool *free_value*)

Destroys a key-value pair.

Parameters

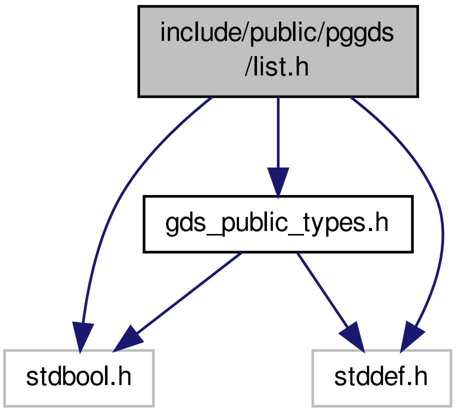
pair	A pointer to the pair to destroy.
free_value	If true, the data will be passed to gdt_free()

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

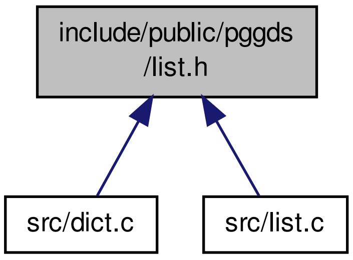
Interface to generic list data structure.

```
#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 ∗ ListItr

  Opaque list iterator type definition.

Functions

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

  Creates a new list.
- void list_destroy (List list)

  Destroys a list.
- bool list_append (List list,...)

  Appends a value to the back of a list.
- bool list_prepend (List list,...)

  Prepends a value to the front of a list.
- bool list_insert (List list, const size_t index,...)

  Inserts a value into a list.
- bool list_delete_front (List list)

  Deletes the value at the front of the list.
- bool list_delete_back (List list)

  Deletes the value at the back of the list.
- bool list_delete_index (List list, const size_t index)

  Deletes the value at the specified index of the list.
- bool list_element_at_index (List list, const size_t index, void ∗p)

  Gets the value at the specified index of the list.
- bool list_set_element_at_index (List list, const size_t index,...)

  Sets the value at the specified index of the list.
- bool list_find (List list, size_t ∗index,...)

  Tests if a value is contained in a list.
- ListItr list_find_itr (List list,...)

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

Sorts a list in-place, in ascending order.

- bool list_reverse_sort (List list)

  Sorts a list in-place, in descending order.

- ListItr list_itr_first (List list)

  Returns an iterator to the first element of the list.

- ListItr list_itr_last (List list)

  Returns an iterator to the last element of the list.

- ListItr list_itr_next (ListItr itr)

  Increments a list iterator.

- ListItr list_itr_previous (ListItr itr)

  Decrements a list iterator.

- void list_get_value_itr (ListItr itr, void ∗p)

  Retrieves a value from an iterator.

- bool list_is_empty (List list)

  Tests if a list is empty.

- size_t list_length (List list)

  Returns the length of a list.

8.25.1 Detailed Description

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

Author

Paul Griffiths

Copyright

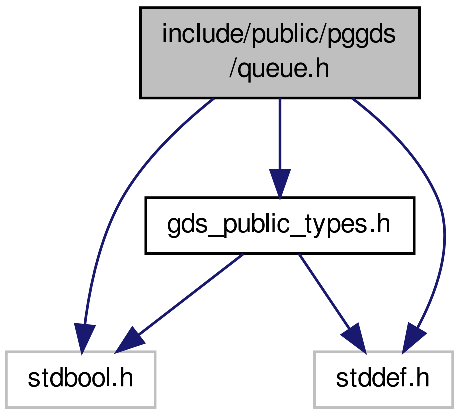
Copyright 2014 Paul Griffiths. Distributed under the terms of the GNU General Public License. http-://www.gnu.org/licenses/

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

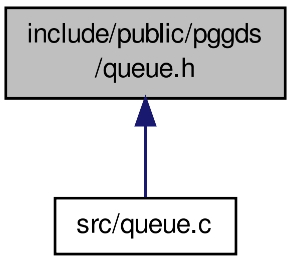
Interface to generic queue data structure.

```
#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

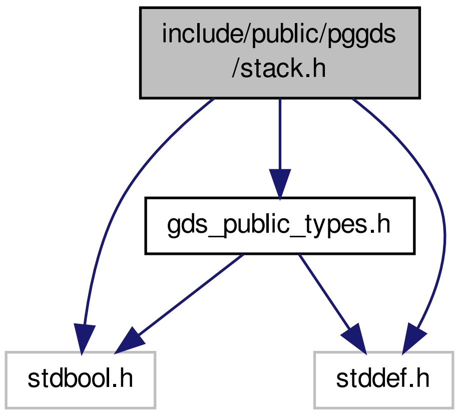
Copyright 2014 Paul Griffiths. Distributed under the terms of the GNU General Public License. `http://www.gnu.org/licenses/`

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

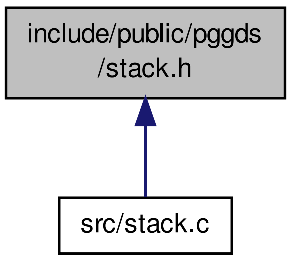
Interface to generic stack data structure.

```
#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

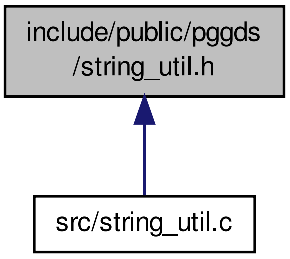
Paul Griffiths

Copyright

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

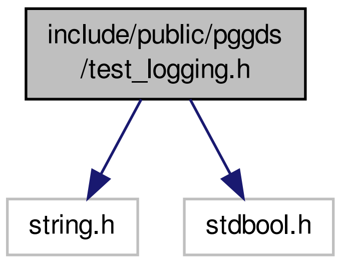
Copyright

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

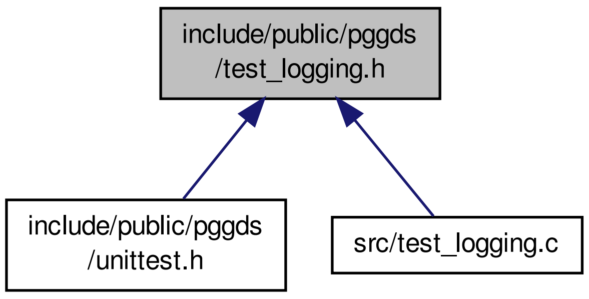
Interface to unit test logging functionality.

```
#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 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.

8.29.1 Detailed Description

Interface to unit test logging functionality.

Author

Paul Griffiths

Copyright

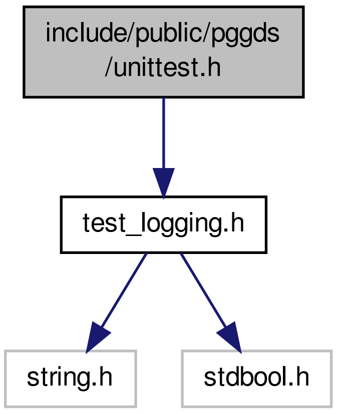
Copyright 2014 Paul Griffiths. Distributed under the terms of the GNU General Public License. http://www.gnu.org/licenses/

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

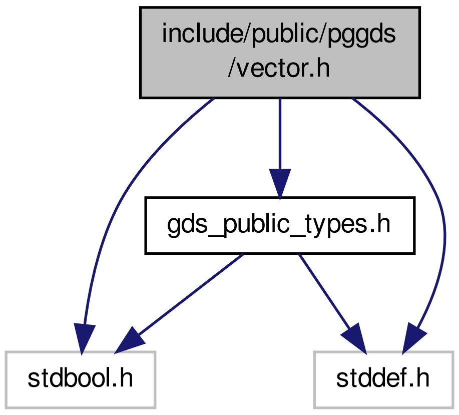
Copyright 2014 Paul Griffiths. Distributed under the terms of the GNU General Public License. `http://www.gnu.org/licenses/`

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

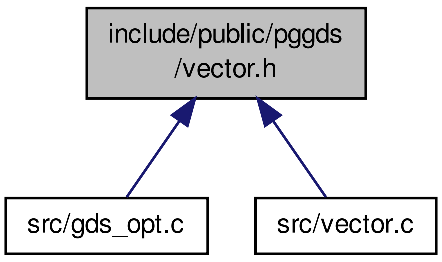
Interface to generic vector data structure.

```
#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

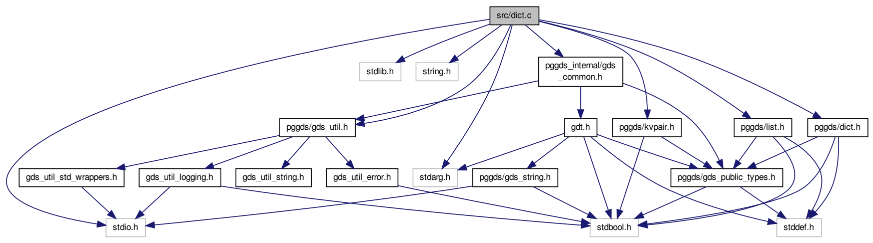
Copyright

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

dict	A pointer to the dictionary.

Return values

true	Success
false	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

dict	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

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

Return values

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

8.32.2.4 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

dict	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

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

Return values

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

8.32.2.6 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

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

Return values

true	Key was found
false	Key was not found

8.32.2.7 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

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

Return values

true	Success
false	Failure, dynamic memory allocation failed

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

Retrieves the value for a key in the dictionary.

Parameters

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

Return values

true	Success
false	Failure, key was not found

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

Calculates a hash of a string.

Uses Dan Bernstein's djb2 algorithm.

Parameters

str	A pointer to a string

Returns

The hash value

8.32.3 Variable Documentation

8.32.3.1 const size_t BUCKETS = 256 `[static]`

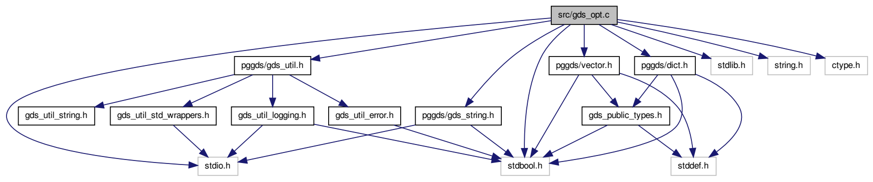
Number of buckets

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

allowed	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]`

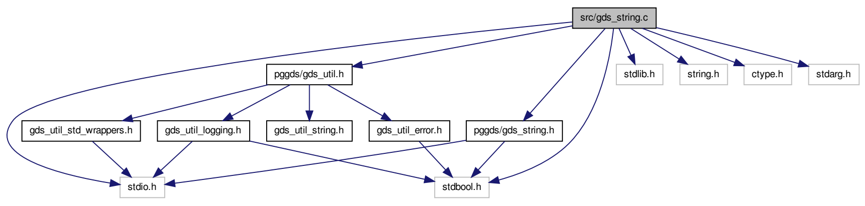
File scope variable to hold program name

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 GDSString

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 (GDSString s1, const char ∗s2)

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

  Returns index of first occurence of a character.
- GDSString gds_str_substr_left (GDSString str, const size_t numchars)

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

  Returns a right substring.
- void gds_str_split (GDSString src, GDSString ∗left, GDSString ∗right, const char sc)

  Splits a string.
- void gds_str_trim_leading (GDSString str)

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

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

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

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

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

  Checks is a string contains only alphanumeric characters.
- void gds_str_clear (GDSString str)

  Clears (empties) a string.
- bool gds_str_intval (GDSString str, const int base, int ∗value)

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

  Gets the double value of a string.
- GDSString gds_str_getline_assign (GDSString str, const size_t size, FILE ∗fp)

  Gets a line from a file and assigns it to 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_decorate (GDSString str, GDSString left_dec, GDSString right_dec)

  Brackets a string with decoration strings.

8.34.1 Detailed Description

Implementation of string data structure.

Author

Paul Griffiths

Copyright

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

str	The string.
new_capacity	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 strings capacity is increased to the required capacity.

Parameters

str	The string.
required_-capacity	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

src	The string to duplicate.
length	A pointer to a `size_t` 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 `strlen()`. This argument is ignored if set to `NULL`.

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

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

Returns

`dst`.

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

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

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

Parameters

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

Returns

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

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

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

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

Parameters

dst	The destination string.
src	The C-style string to concentate with `dst`.
src_length	The length of `src`, 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

str	The string.
numchars	The number of characters to remove.

8.34.2.9 static void gds_str_remove_right (GDSString *str,* const size_t *numchars*) [static]

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

Parameters

str	The string.
numchars	The number of characters to remove.

8.34.2.10 static void truncate_if_needed (GDSString *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.

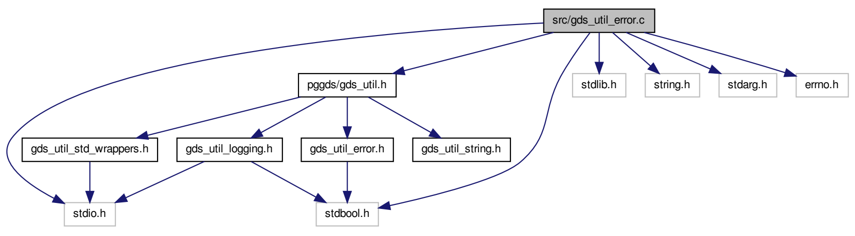
Parameters

str	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 ∗progname, 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.

8.35.1 Detailed Description

Implementation of general utility error functions.

Author

Paul Griffiths

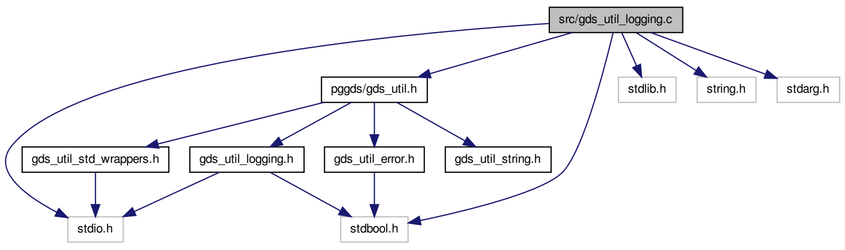
Copyright

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 ∗logfilename, 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]

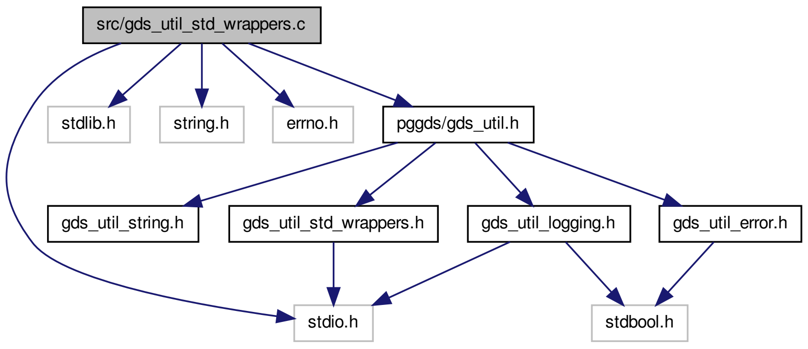
File scope variable for current logging status

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 <pggds/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

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

nmemb	The number of members to allocate.
size	The size in bytes of each member.
filename	The name of the calling file.
linenum	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

path	The path of the file to open.
mode	The mode under which to open the file.
filename	The name of the calling file.
linenum	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

size	The number of bytes to allocate.
filename	The name of the calling file.
linenum	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

ptr	A pointer to the memory to reallocate.
size	The number of bytes for the new allocation.
filename	The name of the calling file.
linenum	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

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

Returns

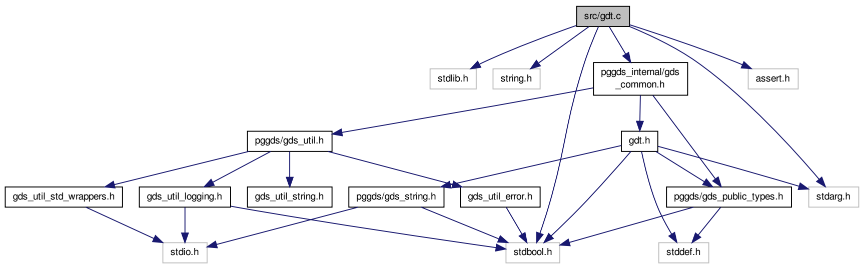
A pointer to the allocated memory.

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

p1	Pointer to first value
p2	Pointer to second value

Return values

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

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

Compare function for double.

Parameters

p1	Pointer to first value
p2	Pointer to second value

Return values

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

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

Compare function for GDSString.

Parameters

p1	Pointer to first value
p2	Pointer to second value

Return values

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

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

Compare function for int.

Parameters

p1	Pointer to first value
p2	Pointer to second value

Return values

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

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

Compare function for long.

Parameters

p1	Pointer to first value
p2	Pointer to second value

Return values

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

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

Compare function for long long.

Parameters

p1	Pointer to first value
p2	Pointer to second value

Return values

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

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

Compare function for signed char.

Parameters

p1	Pointer to first value
p2	Pointer to second value

Return values

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

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

Compare function for size_t.

Parameters

p1	Pointer to first value
p2	Pointer to second value

Return values

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

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

Compare function for string.

Parameters

p1	Pointer to first value
p2	Pointer to second value

Return values

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

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

Compare function for unsigned char.

Parameters

p1	Pointer to first value
p2	Pointer to second value

Return values

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

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

Compare function for unsigned int.

Parameters

p1	Pointer to first value
p2	Pointer to second value

Return values

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

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

Compare function for unsigned long.

Parameters

p1	Pointer to first value
p2	Pointer to second value

Return values

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

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

Compare function for unsigned long long.

Parameters

p1	Pointer to first value
p2	Pointer to second value

Return values

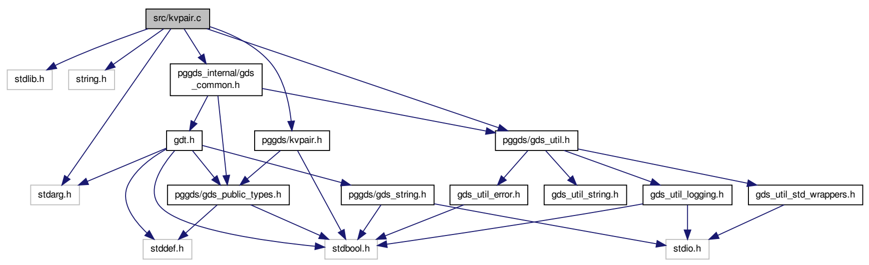
0	First value is equal to second value
-1	First value is less than second value
1	First value is greater than second value

8.39 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

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

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

Return values

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

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

Creates a new key-value pair.

Parameters

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

Return values

NULL	Failure, dynamic memory allocation failed
non-NULL	Success

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

Destroys a key-value pair.

Parameters

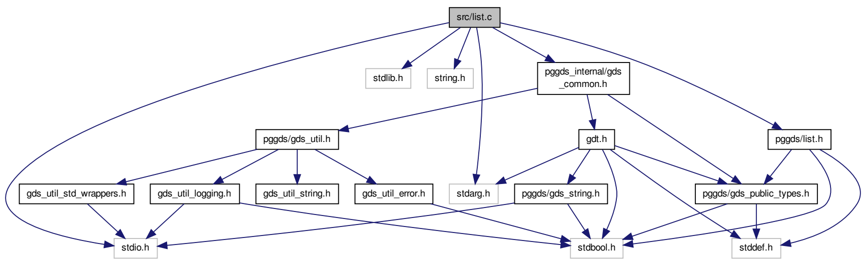
pair	A pointer to the pair to destroy.
free_value	If true, the data will be passed to gdt_free()

8.40 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

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

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

Return values

true	Success
false	Failure, index out of range

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

Private function to return the node at a specified index.

Parameters

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

Return values

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

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

Private function to create list node.

Parameters

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

Return values

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

8.40.3.4 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

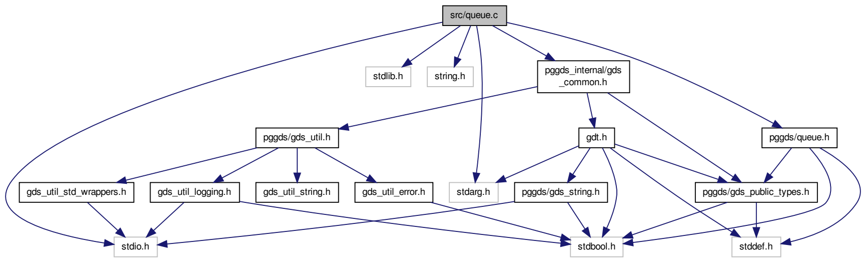
list	A pointer to the list.
node	A pointer to the node.

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 <pggds_internal/gds_common.h>
#include <pggds/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

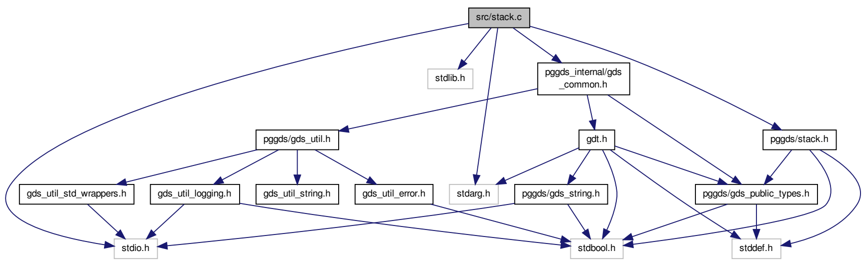
queue_push() relies on this being at least 2.

8.42 src/stack.c File Reference

Implementation of generic stack data structure.

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

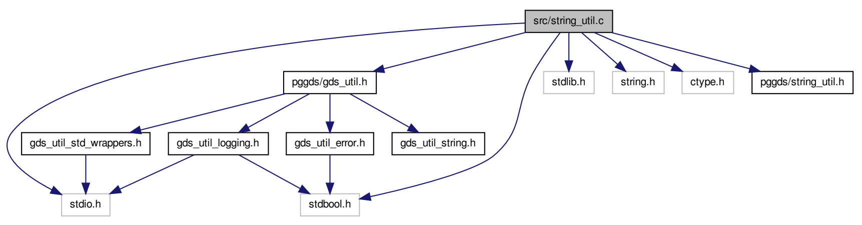
Growth factor for dynamic memory allocation

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

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

Return values

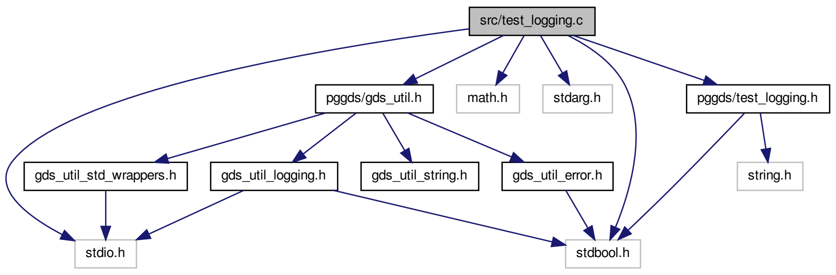
false	Failure, dynamic memory reallocation failed.
true	Success.

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

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

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

success	`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]`

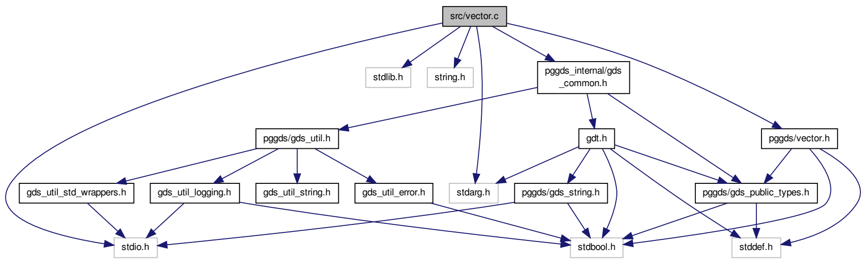
Total number of tests

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

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

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

Return values

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

8.45.3 Variable Documentation

8.45.3.1 const size_t GROWTH = 2 [static]

Growth factor for dynamic memory allocation

Index