LibGGZ Reference Manual

0.0.14

Generated by Doxygen 1.5.1

Fri Nov 30 14:58:02 2007

Contents

1 LibGGZ Module Index **1**

1.1 LibGGZ Modules 1

2 LibGGZ Data Structure Index **3**

2.1 LibGGZ Data Structures 3

3 LibGGZ File Index **5**

3.1 LibGGZ File List 5

4 LibGGZ Page Index **7**

4.1 LibGGZ Related Pages 7

5 LibGGZ Module Documentation **9**

5.1 Memory Handling 9

5.2 Configuration file parsing 14

5.3 List functions 22

5.4 Stacks 30

5.5 XML parsing 32

5.6 Debug/error logging 36

5.7 Miscellaneous convenience functions 41

5.8 Easysock IO 45

5.9 Security functions 60

6 LibGGZ Data Structure Documentation **67**

6.1 _GGZList Struct Reference 67

6.2 _GGZListEntry Struct Reference 69

6.3 _GGZXMLElement Struct Reference 70

6.4 hash_t Struct Reference 71

7 LibGGZ File Documentation **73**

7.1 ggz.h File Reference 73

8 LibGGZ Page Documentation **85**

8.1 Bug List 85

8.2 Todo List 86

Chapter 1

LibGGZ Module Index

1.1 LibGGZ Modules

Here is a list of all modules:

Memory Handling 9
Configuration file parsing 14
List functions 22
Stacks 30
XML parsing 32
Debug/error logging 36
Miscellaneous convenience functions 41
Easysock IO 45
Security functions 60

Chapter 2

LibGGZ Data Structure Index

2.1 LibGGZ Data Structures

Here are the data structures with brief descriptions:

_GGZList (Simple doubly-linked list) 67
_GGZListEntry (A single entry in a **GGZList** (p. 24)) 69
_GGZXMLElement (Object representing a single XML element) 70
hash_t (Hash data structure) 71

Chapter 3

LibGGZ File Index

3.1 LibGGZ File List

Here is a list of all documented files with brief descriptions:

ggz.h (
Author:
Brent M
) 73

Chapter 4

LibGGZ Page Index

4.1 LibGGZ Related Pages

Here is a list of all related documentation pages:

Bug List 85
Todo List 86

Chapter 5

LibGGZ Module Documentation

5.1 Memory Handling

These macros provide an alternative to the normal C library functions for dynamically allocating memory.

Defines

- #define **GGZ_MEM_DEBUG** "ggz_mem"

  Debugging type for memory debugging.

- #define **ggz_malloc**(size) _ggz_malloc(size, _GGZFUNCTION_ " in " __FILE__, __LINE__)

  Macro for memory allocation.

- #define **ggz_realloc**(mem, size) _ggz_realloc(mem, size, _GGZFUNCTION_ " in " __FILE__, __LINE__)

  Macro for resizing previously allocated memory.

- #define **ggz_free**(mem) _ggz_free(mem, _GGZFUNCTION_ " in " __FILE__, __LINE__)

  Macro for freeing memory previously allocated.

- #define **ggz_strdup**(string) _ggz_strdup(string, _GGZFUNCTION_ " in " __FILE__, __LINE__)

  Macro for duplicating string.

Functions

- void * **_ggz_malloc** (const size_t size, const char *tag, int line)

  Function to actually perform memory allocation.

- void * **_ggz_realloc** (const void *ptr, const size_t size, const char *tag, int line) ggz__attribute((warn_unused_result))

Function to perform memory reallocation.

- int **_ggz_free** (const void *ptr, const char *tag, int line)

  Function to free allocated memory.

- char * **_ggz_strdup** (const char *ptr, const char *tag, int line) ggz__attribute((warn_unused_result))

  Function to copy a string.

- char * **ggz_strncpy** (char *dst, const char *src, size_t n)

  Safe version of strncpy.

- int **ggz_memory_check** (void)

  Check memory allocated against memory freed and display any discrepancies.

5.1.1 Detailed Description

These macros provide an alternative to the normal C library functions for dynamically allocating memory.

They keep track of memory allocated by storing the name of the function and file in which they were called.

You can then call **ggz_memory_check()** (p. 13) to make sure all allocated memory has been freed. Note that you will need to enable MEMORY debugging to see this.

5.1.2 Define Documentation

5.1.2.1 #define GGZ_MEM_DEBUG "ggz_mem"

Debugging type for memory debugging.

See also:

ggz_debug_enable (p. 38)

5.1.2.2 #define ggz_malloc(size) _ggz_malloc(size, __GGZFUNCTION__ " in " __FILE__, __LINE__)

Macro for memory allocation.

Parameters:

size the size of memory to allocate, in bytes

Returns:

a pointer to the newly allocated and zeroed memory

5.1.2.3 #define ggz_realloc(mem, size) _ggz_realloc(mem, size, _GGZFUNCTION_ " in " __FILE__, __LINE__)

Macro for resizing previously allocated memory.

Parameters:

mem pointer to memory to reallocate

size new size requested, in bytes

Returns:

pointer to allocated memory

5.1.2.4 #define ggz_free(mem) _ggz_free(mem, _GGZFUNCTION_ " in " __FILE__, __LINE__)

Macro for freeing memory previously allocated.

Parameters:

mem pointer to allocated memory

Returns:

failure code

5.1.2.5 #define ggz_strdup(string) _ggz_strdup(string, _GGZFUNCTION_ " in " __FILE__, __LINE__)

Macro for duplicating string.

Parameters:

string string to duplicate

Returns:

pointer to new string

Note:

It is safe to pass a NULL string.

5.1.3 Function Documentation

5.1.3.1 void* _ggz_malloc (const size_t *size*, const char * *tag*, int *line*)

Function to actually perform memory allocation.

Don't call this directly. Instead, call **ggz_malloc()** (p. 10).

Parameters:

size size of memory to allocate, in bytes

tag string describing the calling function

line linenumber

Returns:

pointer to newly allocated, zeroed memory

5.1.3.2 void* _ggz_realloc (const void * *ptr*, const size_t *size*, const char * *tag*, int *line*)

Function to perform memory reallocation.

Don't call this directly. Instead, call **ggz_realloc()** (p. 11).

Parameters:

ptr pointer to memory to reallocate

size new size, in bytes

tag string describing the calling function

line linenumber

Returns:

pointer to allocated memory

5.1.3.3 int _ggz_free (const void * *ptr*, const char * *tag*, int *line*)

Function to free allocated memory.

Don't call this directly. Instead, call **ggz_free()** (p. 11).

Parameters:

ptr pointer to memory

tag string describing the calling function

line linenumber

Returns:

0 on success, -1 on error

5.1.3.4 char* _ggz_strdup (const char * *ptr*, const char * *tag*, int *line*)

Function to copy a string.

Don't call this directly. Instead, call **ggz_strdup()** (p. 11).

Parameters:

ptr string to duplicate

tag string describing the calling function

line linenumber

Returns:

newly allocated string

Note:

It is safe to pass a NULL string.

5.1.3.5 char* ggz_strncpy (char * *dst*, const char * *src*, size_t *n*)

Safe version of strncpy.

This function will behave like strncpy(), except that the result string is always guaranteed to be NULL-terminated. This matches the behaviour of snprintf() and leads to generally safer code.

Parameters:

dst destination string buffer

src source string

n number of bytes to copy, should be less than buffer size

Returns:

destination buffer

Note:

Passing a NULL string will result in an empty string

5.1.3.6 int ggz_memory_check (void)

Check memory allocated against memory freed and display any discrepancies.

Returns:

0 if no allocated memory remains, -1 otherwise

5.2 Configuration file parsing

Configuration file routines to store and retrieve values.

Defines

- #define **GGZ_CONF_DEBUG** "ggz_conf"

  Debugging type for config-file debugging.

Enumerations

- enum **GGZConfType** { **GGZ_CONF_RDONLY** = ((unsigned char) 0x01), **GGZ_CONF_RDWR** = ((unsigned char) 0x02), **GGZ_CONF_CREATE** = ((unsigned char) 0x04) }

  Specifies the mode for opening a configuration file.

Functions

- void **ggz_conf_cleanup** (void)

  Closes all open configuration files.

- void **ggz_conf_close** (int handle)

  Closes one configuration file.

- int **ggz_conf_parse** (const char *path, const **GGZConfType** options)

  Opens a configuration file and parses the variables so they can be retrieved with the access functions.

- int **ggz_conf_commit** (int handle)

  Commits any changed variables to the configuration file.

- int **ggz_conf_write_string** (int handle, const char *section, const char *key, const char *value)

  Writes a string value to a section and key in an open configuration file.

- int **ggz_conf_write_int** (int handle, const char *section, const char *key, int value)

  Writes an integer value to a section and key in an open configuration file.

- int **ggz_conf_write_list** (int handle, const char *section, const char *key, int argc, char **argv)

  Writes a list of string values to a section and key in an open configuration file.

- char * **ggz_conf_read_string** (int handle, const char *section, const char *key, const char *def)

  Reads a string value from an open configuration file.

- int **ggz_conf_read_int** (int handle, const char *section, const char *key, int def)

Reads an integer value from an open configuration file.

- int **ggz_conf_read_list** (int handle, const char *section, const char *key, int *argcp, char ***argvp)

  Reads a list of string values from an open configuration file.

- int **ggz_conf_remove_section** (int handle, const char *section)

  This will remove an entire section and all its associated keys from a configuration file.

- int **ggz_conf_remove_key** (int handle, const char *section, const char *key)

  This will remove a single key from a configuration file.

- int **ggz_conf_get_sections** (int handle, int *argcp, char ***argvp)

  This function returns a list of all sections in a config file.

- int **ggz_conf_get_keys** (int handle, const char *section, int *argcp, char ***argvp)

  This function returns a list of all keys within a section in a config file.

5.2.1 Detailed Description

Configuration file routines to store and retrieve values.

Configuration file parsing begins by calling **ggz_conf_parse()** (p. 17) to open a config file. The file can be created automatically if GGZ_CONF_CREATE is specified. The returned handle uniquely identifies the configuration file, so multiple files can be open at one time.

If the same configuration file is opened multiple times, the original handle will be returned and only one copy will be retained within memory. One use of this feature is that a file may be opened read only initially and later have writing enabled by merely reparsing the file using the same pathname. Note that this feature can be fooled if the same file is opened using different pathnames. Chaos may or may not result in this case, and it is considered a feature not a bug.

Values are stored using a system of section and keys. A key must be unique within a section (you cannot store both an integer and a string under the same key. Section and key names may contain any characters (including whitespace) other than an equals sign. Although keys may not have leading or trailing whitespace, section names may. It is suggested that any whitespace (other than possibly internal spaces) be avoided when specifying section and key names. Alphabetic case is preserved, and must be matched.

An important caveat to remember when using the configuration functions is that writing a value only caches the value in memory. In order to write the values to the physical file, **ggz_conf_commit()** (p. 17) must be called at some point. This makes writing multiple values in rapid succession more efficient, as the entire file must be regenerated in order to be written to the flat-file format of the configuration file.

The string and list reading functions return dynamically allocated memory to the caller. The user is responsible for calling **ggz_free()** (p. 11) on this memory when they no longer need the returned values.

All memory used internally by the configuration functions will be released when **ggz_conf_cleanup()** (p. 16) is called. Note that this does NOT commit any changes made to the configuration files. The user is expected to call this at program termination, but it may be called at any time earlier than termination and new files may be subsequently opened.

Bug

The maximum length of a configuration file line is fixed at 1024 characters. Exceeding this length will result in parsing errors when the file is parsed. No internal problems should (hopefully) result, but values will be lost.

5.2.2 Define Documentation

5.2.2.1 #define GGZ_CONF_DEBUG "ggz_conf"

Debugging type for config-file debugging.

See also:

ggz_debug_enable (p. 38)

5.2.3 Enumeration Type Documentation

5.2.3.1 enum GGZConfType

Specifies the mode for opening a configuration file.

See also:

ggz_conf_parse() (p. 17)

Enumerator:

GGZ_CONF_RDONLY Read only.
GGZ_CONF_RDWR Read and write.
GGZ_CONF_CREATE Create file.

5.2.4 Function Documentation

5.2.4.1 void ggz_conf_cleanup (void)

Closes all open configuration files.

Note:

This does not automatically commit changed values. Any non-committed values written will be lost.

5.2.4.2 void ggz_conf_close (int *handle*)

Closes one configuration file.

Note:

The same warning as for **ggz_conf_cleanup()** (p. 16) applies here.

5.2.4.3 int ggz_conf_parse (const char * *path*, const GGZConfType *options*)

Opens a configuration file and parses the variables so they can be retrieved with the access functions.

Parameters:

path A string specifying the filename to be parsed

options An or'ed set of GGZConfType option bits

Returns:

An integer configuration file handle or -1 on error

See also:

GGZConfType (p. 16)

5.2.4.4 int ggz_conf_commit (int *handle*)

Commits any changed variables to the configuration file.

The configuration file remains open and may continue to be written to.

Parameters:

handle A valid handle returned by **ggz_conf_parse()** (p. 17)

Returns:

0 if successful, -1 on failure

5.2.4.5 int ggz_conf_write_string (int *handle*, const char * *section*, const char * *key*, const char * *value*)

Writes a string value to a section and key in an open configuration file.

Parameters:

handle A valid handle returned by **ggz_conf_parse()** (p. 17)

section A string section name to write to

key A string variable key name to write to

value The string value to write

Returns:

0 if successful, -1 on failure

5.2.4.6 int ggz_conf_write_int (int *handle*, const char * *section*, const char * *key*, int *value*)

Writes an integer value to a section and key in an open configuration file.

Parameters:

handle A valid handle returned by **ggz_conf_parse()** (p. 17)

section A string section name to write to

key A string variable key name to write to

value The integer value to write

Returns:

0 if successful, -1 on failure

5.2.4.7 int ggz_conf_write_list (int *handle*, const char * *section*, const char * *key*, int *argc*, char ** *argv*)

Writes a list of string values to a section and key in an open configuration file.

Parameters:

handle A valid handle returned by **ggz_conf_parse()** (p. 17)

section A string section name to write to

key A string variable key name to write to

argc The number of string array entries in argv

argv An array of strings to create a list from

Returns:

0 if successful, -1 on failure

5.2.4.8 char* ggz_conf_read_string (int *handle*, const char * *section*, const char * *key*, const char * *def*)

Reads a string value from an open configuration file.

If the section/key combination is not found, the default entry is returned.

Parameters:

handle A valid handle returned by **ggz_conf_parse()** (p. 17)

section A string section name to read from

key A string variable key name to read from

def A value to be returned if the entry does not exist (may be NULL)

Returns:

A dynamically allocated copy of the stored (or default) value or NULL

Note:

The copy is allocated via a standard **ggz_malloc()** (p. 10) call and the caller is expected to be responsible for calling **ggz_free()** (p. 11) on the returned value when they no longer need the value. No memory is allocated if a default value of NULL is returned.

5.2.4.9 int ggz_conf_read_int (int *handle*, const char * *section*, const char * *key*, int *def*)

Reads an integer value from an open configuration file.

If the section/key combination is not found, the default entry is returned.

Parameters:

handle A valid handle returned by **ggz_conf_parse()** (p. 17)

section A string section name to read from

key A string variable key name to read from

def A value to be returned if the entry does not exist

Returns:

The integer value stored in the configuration file, or the default

5.2.4.10 int ggz_conf_read_list (int *handle*, const char * *section*, const char * *key*, int * *argcp*, char *** *argvp*)

Reads a list of string values from an open configuration file.

Parameters:

handle A valid handle returned by **ggz_conf_parse()** (p. 17)

section A string section name to read from

key A string variable key name to read from

argcp A pointer to an integer which will receive the number of list entries read

argvp A pointer to a string array. This will receive a value pointing to a dynamically allocated array of string values. The list will be NULL-terminated.

Returns:

0 on success, -1 on failure

Note:

The array is allocated via standard **ggz_malloc()** (p. 10) calls and the caller is expected to be responsible for calling **ggz_free()** (p. 11) on the string values and the associated array structure when they no longer need the list. If the section/key combination is not found -1 will be returned, *argcp is set to a value of zero, no memory will be allocated, and *argvp will be set to NULL.

5.2.4.11 int ggz_conf_remove_section (int *handle*, const char * *section*)

This will remove an entire section and all its associated keys from a configuration file.

Parameters:

handle A valid handle returned by **ggz_conf_parse()** (p. 17)

section A string section name to remove

Returns:

0 on success, -1 on failure

5.2.4.12 int ggz_conf_remove_key (int *handle*, const char * *section*, const char * *key*)

This will remove a single key from a configuration file.

Parameters:

handle A valid handle returned by **ggz_conf_parse()** (p. 17)

section A string section name the key is located in

key A string key name to remove

Returns:

0 on success, -1 on failure

5.2.4.13 int ggz_conf_get_sections (int *handle*, int * *argcp*, char *** *argvp*)

This function returns a list of all sections in a config file.

Parameters:

handle A valid handle returned by **ggz_conf_parse()** (p. 17)

argcp A pointer to an integer which will receive the number of sections in the configuration file

argvp A pointer to a string array. This will receive a value pointing to a dynamically allocated array of string values. This list is NOT NULL terminated.

Returns:

0 on success, -1 on failure

Note:

The array is allocated via standard **ggz_malloc()** (p. 10) calls and the caller is expected to be responsible for calling **ggz_free()** (p. 11) on the string values and the associated array structure when they no longer need the list.

5.2.4.14 int ggz_conf_get_keys (int *handle*, const char * *section*, int * *argcp*, char *** *argvp*)

This function returns a list of all keys within a section in a config file.

Parameters:

handle A valid handle returned by **ggz_conf_parse()** (p. 17)

section A string section name

argcp A pointer to an integer which will receive the number of lists within section in the configuration file

argvp A pointer to a string array. This will receive a value pointing to a dynamically allocated array of string values. This list is NOT NULL terminated.

Returns:

0 on success, -1 on failure

Note:

The array is allocated via standard **ggz_malloc()** (p. 10) calls and the caller is expected to be responsible for calling **ggz_free()** (p. 11) on the string values and the associated array structure when they no longer need the list.

5.3 List functions

Data Structures and functions for manipulating linked-lists.

Data Structures

- struct **_GGZListEntry**
  *A single entry in a **GGZList** (p. 24).*
- struct **_GGZList**
  Simple doubly-linked list.

Defines

- #define **GGZ_LIST_REPLACE_DUPS** 0x00
  Overwrite duplicate values on insert.
- #define **GGZ_LIST_ALLOW_DUPS** 0x01
  Allow duplicate data entries to exist in the list.

Typedefs

- typedef int(*) **ggzEntryCompare** (const void *a, const void *b)
  *A function type for doing data comparison on two items in a **GGZList** (p. 24).*
- typedef void *(*) **ggzEntryCreate** (void *data)
  *A function type for creating a copy of a data item for insertion into a **GGZList** (p. 24).*
- typedef void(*) **ggzEntryDestroy** (void *data)
  *A function type to destroy an entry in a **GGZList** (p. 24).*
- typedef **_GGZListEntry GGZListEntry**
  *A single entry in a **GGZList** (p. 24).*
- typedef **_GGZList GGZList**
  Simple doubly-linked list.

Functions

- **GGZList** * **ggz_list_create** (**ggzEntryCompare** compare_func, **ggzEntryCreate** create_func, **ggzEntryDestroy** destroy_func, int options)
  *Create a new **GGZList** (p. 24).*
- int **ggz_list_insert** (**GGZList** *list, void *data)
  Insert data into a list.

- **GGZListEntry** * **ggz_list_head** (**GGZList** *list)

  Get the first node of a list.

- **GGZListEntry** * **ggz_list_tail** (**GGZList** *list)

  Get the last node of a list.

- **GGZListEntry** * **ggz_list_next** (**GGZListEntry** *entry)

  Get the next node of a list.

- **GGZListEntry** * **ggz_list_prev** (**GGZListEntry** *entry)

  Get the previous node of a list.

- void * **ggz_list_get_data** (**GGZListEntry** *entry)

  Retrieve the data stored in a list entry.

- **GGZListEntry** * **ggz_list_search** (**GGZList** *list, void *data)

  Search for a specified data item in the list.

- **GGZListEntry** * **ggz_list_search_alt** (**GGZList** *list, void *data, **ggzEntryCompare** compare_func)

  Search for a specified data item in the list using a provided comparison function.

- void **ggz_list_delete_entry** (**GGZList** *list, **GGZListEntry** *entry)

  Removes an entry from a list, calling a destructor if registered.

- void **ggz_list_free** (**GGZList** *list)

  Free all resources associated with a list.

- int **ggz_list_count** (**GGZList** *list)

  Get the length of the list.

- int **ggz_list_compare_str** (void *a, void *b)

  Compare two character strings.

- void * **ggz_list_create_str** (void *data)

  Copy a character string.

- void **ggz_list_destroy_str** (void *data)

  Free a character string.

5.3.1 Detailed Description

Data Structures and functions for manipulating linked-lists.

5.3.2 Typedef Documentation

5.3.2.1 typedef int(∗) ggzEntryCompare(const void ∗a, const void ∗b)

A function type for doing data comparison on two items in a **GGZList** (p. 24).

Parameters:

a An arbitrary element in the **GGZList** (p. 24).

b An arbitrary element in the **GGZList** (p. 24).

Returns:

Negative if a < b; 0 if a == b; positive if a > b.

5.3.2.2 typedef void∗(∗) ggzEntryCreate(void ∗data)

A function type for creating a copy of a data item for insertion into a **GGZList** (p. 24).

A function of this type may be called on an element when it is first inserted into a **GGZList** (p. 24).

Parameters:

data A pointer to the data given to the list for insertion

Returns:

A new copy of the data, safe for list insertion

5.3.2.3 typedef void(∗) ggzEntryDestroy(void ∗data)

A function type to destroy an entry in a **GGZList** (p. 24).

A function of this type may be called on an element when it is removed from a **GGZList** (p. 24).

Parameters:

data The entry being removed

5.3.2.4 typedef struct _GGZListEntry GGZListEntry

A single entry in a **GGZList** (p. 24).

Do not access these members directly, but use the **ggz_list_get_data()** (p. 27), **ggz_list_next()** (p. 26) and **ggz_list_prev()** (p. 26) accessor functions instead.

5.3.2.5 typedef struct _GGZList GGZList

Simple doubly-linked list.

Do not access these members directly. Instead use accessor functions.

5.3.3 Function Documentation

5.3.3.1 GGZList* ggz_list_create (ggzEntryCompare *compare_func*, ggzEntryCreate *create_func*, ggzEntryDestroy *destroy_func*, int *options*)

Create a new **GGZList** (p. 24).

When creating a a new list, you have some control over its behavior. The first parameter, compare_func allows you to specify a comparison for sorting the list elements. If you specify NULL for compare_func, the list will be unordered. The second parameter, create_func allows you to specify how new copies of data will be created during insertion. **GGZList** (p. 24) objects stores pointers to your data in **GGZListEntry** (p. 24) nodes. If you specify NULL for create_func, the list will store the actual pointer to your data when you call **ggz_list_insert()** (p. 25). If you specify a create_func, it will be called to create a new copy of the object for storage in the list. You are then safely deallocate the original copy of the data. The third parameter, destroy_func allows you to specify a deallocation function for data entries when they are removed from the list.

Note:

The functions **ggz_list_compare_str()** (p. 28), **ggz_list_create_str()** (p. 29), **ggz_list_destroy_str()** (p. 29) are provided for use with character string data.

The last argument must be one of **GGZ_LIST_REPLACE_DUPS** (p. 74) or **GGZ_LIST_ALLOW_DUPS** (p. 74), to specify how the list will behave with respect to duplicate data entries. If **GGZ_LIST_REPLACE_DUPS** (p. 74) is passed, duplicate entries (as determined by compare_func) will be replaced upon **ggz_list_insert()** (p. 25). If **GGZ_LIST_ALLOW_DUPS** (p. 74) is specified, duplicate data entries will be allowed to exist in the list.

Note:

If compare_func is NULL, **GGZ_LIST_REPLACE_DUPS** (p. 74) has no meaning.

Parameters:

compare_func Function to use for comparing data items

create_func Function to use for allocating and copying data items

destroy_func Function to use for dellocating data items

options One of **GGZ_LIST_REPLACE_DUPS** (p. 74) or **GGZ_LIST_ALLOW_DUPS** (p. 74) specifying how the list should handle duplicate data entries

Returns:

A pointer to a newly allocated **GGZList** (p. 24)

5.3.3.2 int ggz_list_insert (GGZList * *list*, void * *data*)

Insert data into a list.

Parameters:

list Pointer to a **GGZList** (p. 24)

data Pointer to data to be inserted

Returns:

-1 on failure, 0 is the item was inserted, and 1 if the item replaced an existing list item

Note:

Replacement of duplicate items only occurs if **GGZ_LIST_REPLACE_DUPS** (p. 74) was passed to **ggz_list_create()** (p. 25).

5.3.3.3 GGZListEntry* ggz_list_head (GGZList * *list*)

Get the first node of a list.

Parameters:

list Pointer to a **GGZList** (p. 24)

Returns:

The **GGZListEntry** (p. 24) that is first in the list

5.3.3.4 GGZListEntry* ggz_list_tail (GGZList * *list*)

Get the last node of a list.

Parameters:

list Pointer to a **GGZList** (p. 24)

Returns:

The **GGZListEntry** (p. 24) that is last in the list

5.3.3.5 GGZListEntry* ggz_list_next (GGZListEntry * *entry*)

Get the next node of a list.

Parameters:

entry Pointer to a **GGZListEntry** (p. 24) in a **GGZList** (p. 24)

Returns:

The next **GGZListEntry** (p. 24) in the list

5.3.3.6 GGZListEntry* ggz_list_prev (GGZListEntry * *entry*)

Get the previous node of a list.

Parameters:

entry Pointer to a **GGZListEntry** (p. 24) in a **GGZList** (p. 24)

Returns:

The previous **GGZListEntry** (p. 24) in the list

5.3.3.7 void* ggz_list_get_data (GGZListEntry * *entry*)

Retrieve the data stored in a list entry.

Parameters:

entry Pointer to a **GGZListEntry** (p. 24)

Returns:

Pointer to the data stored in the specifed node (**GGZListEntry** (p. 24))

5.3.3.8 GGZListEntry* ggz_list_search (GGZList * *list*, void * *data*)

Search for a specified data item in the list.

ggz_list_search() (p. 27) searches a list for a particular data item using the ggzEntryCompare function provided as compare_func to **ggz_list_create()** (p. 25). If you wish to search using an alternative comparison function, see **ggz_list_search_alt()** (p. 27).

Parameters:

list Pointer to a **GGZList** (p. 24)

data Pointer to data to search for

Returns:

Pointer to the **GGZListEntry** (p. 24) containing the specifed node (NULL if the data could not be found or if no compare_func was specified at list-creation time)

5.3.3.9 GGZListEntry* ggz_list_search_alt (GGZList * *list*, void * *data*, ggzEntryCompare *compare_func*)

Search for a specified data item in the list using a provided comparison function.

ggz_list_search_alt() (p. 27) searches a list for a particular data item using the passed ggzEntryCompare function.

Parameters:

list Pointer to a **GGZList** (p. 24)

data Pointer to data to search for

compare_func Comparison function

Returns:

Pointer to the **GGZListEntry** (p. 24) containing the specified node (NULL if the data could not be found or if no compare_func was specified)

5.3.3.10 void ggz_list_delete_entry (GGZList * *list*, GGZListEntry * *entry*)

Removes an entry from a list, calling a destructor if registered.

ggz_list_delete_entry() (p. 28) removes the specifed entry from the list. If a ggzEntryDestroy function was passed as destroy_func to **ggz_list_create()** (p. 25), it will be called on the data item after it has been removed.

Parameters:

list Pointer to a **GGZList** (p. 24)

entry Pointer to the **GGZListEntry** (p. 24) to remove

5.3.3.11 void ggz_list_free (GGZList * *list*)

Free all resources associated with a list.

ggz_list_free() (p. 28) will free all resources allocated by the list. If a ggzEntryDestroy function was passed as destroy_func to **ggz_list_create()** (p. 25), it will be called on all data items in the list as well.

Parameters:

list Pointer to a **GGZList** (p. 24)

5.3.3.12 int ggz_list_count (GGZList * *list*)

Get the length of the list.

Parameters:

list Pointer to a **GGZList** (p. 24)

Returns:

The number of entries in the list

5.3.3.13 int ggz_list_compare_str (void * *a*, void * *b*)

Compare two character strings.

This function is intended to be passed as the compare_func of **ggz_list_create()** (p. 25) when creating a list that stores character strings.

Parameters:

a A string to compare

b A second string to compare

Returns:

The result of strcmp() on a and b, or 1 if either is NULL

5.3.3.14 void* ggz_list_create_str (void * *data*)

Copy a character string.

This function is intended to be passed as the create_func of **ggz_list_create()** (p. 25) when creating a list that stores character strings.

Parameters:

data A string to copy

Returns:

A newly allocated copy of the string

5.3.3.15 void ggz_list_destroy_str (void * *data*)

Free a character string.

This function is intended to be passed as the destroy_func of **ggz_list_create()** (p. 25) when creating a list that stores character strings.

Parameters:

data The string to deallocate

5.4 Stacks

Data Structures and functions for manipulating stacks.

Typedefs

- typedef **_GGZList GGZStack**
  *Simple implementation of stacks using **GGZList** (p. 24).*

Functions

- **GGZStack** * **ggz_stack_new** (void)
  Create a new stack.
- void **ggz_stack_push** (**GGZStack** *stack, void *data)
  Push a data item onto the top of the stack.
- void * **ggz_stack_pop** (**GGZStack** *stack)
  Pop the top item off of the stack.
- void * **ggz_stack_top** (**GGZStack** *stack)
  Get the top item on the stack without popping it.
- void **ggz_stack_free** (**GGZStack** *stack)
  Free the stack.

5.4.1 Detailed Description

Data Structures and functions for manipulating stacks.

5.4.2 Function Documentation

5.4.2.1 GGZStack* ggz_stack_new (void)

Create a new stack.

Returns:

Pointer to a newly allocated **GGZStack** (p. 74) object

5.4.2.2 void ggz_stack_push (GGZStack * *stack*, void * *data*)

Push a data item onto the top of the stack.

Parameters:

stack Pointer to a **GGZStack** (p. 74)
data Pointer to data to insert onto stack

5.4.2.3 void* ggz_stack_pop (GGZStack * *stack*)

Pop the top item off of the stack.

Parameters:

stack Pointer to a **GGZStack** (p. 74)

Returns:

Pointer to the data item on the top of the stack

5.4.2.4 void* ggz_stack_top (GGZStack * *stack*)

Get the top item on the stack without popping it.

Parameters:

stack Pointer to a **GGZList** (p. 24)

Returns:

Pointer to the data item currently in to of the stack

5.4.2.5 void ggz_stack_free (GGZStack * *stack*)

Free the stack.

Note:

This does not free the data stored in the stack.

Parameters:

stack Pointer to a **GGZStack** (p. 74)

5.5 XML parsing

Utility functions for doing simple XML parsing.

Data Structures

- struct **_GGZXMLElement**
  Object representing a single XML element.

Typedefs

- typedef **_GGZXMLElement GGZXMLElement**
  Object representing a single XML element.

Functions

- **GGZXMLElement** * **ggz_xmlelement_new** (const char *tag, const char *const *attrs, void(*process)(void *, **GGZXMLElement** *), void(*free)(**GGZXMLElement** *))
  *Create a new **GGZXMLElement** (p. 75) element.*

- void **ggz_xmlelement_init** (**GGZXMLElement** *element, const char *tag, const char *const *attrs, void(*process)(void *, **GGZXMLElement** *), void(*free)(**GGZXMLElement** *))
  *Initialize a **GGZXMLElement** (p. 75).*

- void **ggz_xmlelement_set_data** (**GGZXMLElement** *element, void *data)
  *Set ancillary data on a **GGZXMLElement** (p. 75) object.*

- const char * **ggz_xmlelement_get_tag** (**GGZXMLElement** *element)
  Get an XML element's name.

- const char * **ggz_xmlelement_get_attr** (**GGZXMLElement** *element, const char *attr)
  Get the value of an attribute on XML element.

- void * **ggz_xmlelement_get_data** (**GGZXMLElement** *element)
  Get the user-supplied data associated with an XML element.

- char * **ggz_xmlelement_get_text** (**GGZXMLElement** *element)
  Get an XML element's content text.

- void **ggz_xmlelement_add_text** (**GGZXMLElement** *element, const char *text, int len)
  Append a string to the element's content text.

- void **ggz_xmlelement_free** (**GGZXMLElement** *element)
  Free the memory associated with an XML element.

5.5.1 Detailed Description

Utility functions for doing simple XML parsing.

These can be used with streaming XML parsers, and don't have the overhead of a full DOM tree. **GGZXMLElement** (p. 75) represents a single element, along with its attributes and text data.

Note:

This does not parse your XML. It is simply for use to store the data as you are parsing.

5.5.2 Function Documentation

5.5.2.1 GGZXMLElement* ggz_xmlelement_new (const char * *tag*, const char *const * *attrs*, void(*)(void *, GGZXMLElement *) *process*, void(*)(GGZXMLElement *) *free*)

Create a new **GGZXMLElement** (p. 75) element.

Parameters:

tag The name of the XML element (tag)

attrs NULL terminated array of attributes/values. These must alternate: attribute1, value1, attribute2, value2, etc.

process User-defined function for processing XML elements

free User-defined function for deallocating **GGZXMLElement** (p. 75) objects. If provided, this will be invoked by **ggz_xmlelement_free()** (p. 35), and in addition to any user-defined processing should call **ggz_free()** (p. 11) the element itself.

Returns:

Pointer to a newly allocated **GGZXMLElement** (p. 75) object

5.5.2.2 void ggz_xmlelement_init (GGZXMLElement * *element*, const char * *tag*, const char *const * *attrs*, void(*)(void *, GGZXMLElement *) *process*, void(*)(GGZXMLElement *) *free*)

Initialize a **GGZXMLElement** (p. 75).

Parameters:

element Pointer to a **GGZXMLElement** (p. 75) to initialize

tag The name of the XML element (tag)

attrs NULL terminated array of attributes/values. These must alternate: attribute1, value1, attribute2, value2, etc.

process User-defined function for processing XML elements

free User-defined function for deallocating **GGZXMLElement** (p. 75) objects. If provided, this will be invoked by **ggz_xmlelement_free()** (p. 35), and in addition to any user-defined processing should call **ggz_free()** (p. 11) the element itself.

Returns:

Pointer to a newly allocated **GGZXMLElement** (p. 75) object

5.5.2.3 void ggz_xmlelement_set_data (GGZXMLElement * *element*, void * *data*)

Set ancillary data on a **GGZXMLElement** (p. 75) object.

Associate some extra data with an XML element.

Parameters:

element Pointer to an XML element

data Pointer to user-supplied data

Returns:

The element's name

5.5.2.4 const char* ggz_xmlelement_get_tag (GGZXMLElement * *element*)

Get an XML element's name.

Parameters:

element Pointer to an XML element

Returns:

The element's name

5.5.2.5 const char* ggz_xmlelement_get_attr (GGZXMLElement * *element*, const char * *attr*)

Get the value of an attribute on XML element.

Parameters:

element Pointer to an XML element

attr An attribute name

Returns:

The value of the attribute, or NULL is there is no such attribute present

5.5.2.6 void* ggz_xmlelement_get_data (GGZXMLElement * *element*)

Get the user-supplied data associated with an XML element.

Parameters:

element Pointer to an XML element

Returns:

Pointer to the user-supplied data

5.5.2.7 char* ggz_xmlelement_get_text (GGZXMLElement * *element*)

Get an XML element's content text.

Parameters:

element Pointer to an XML element

Returns:

The text content of the element

5.5.2.8 void ggz_xmlelement_add_text (GGZXMLElement * *element*, const char * *text*, int *len*)

Append a string to the element's content text.

Parameters:

element Pointer to an XML element

text String to append

len The string's length, in bytes

5.5.2.9 void ggz_xmlelement_free (GGZXMLElement * *element*)

Free the memory associated with an XML element.

Parameters:

element Pointer to an XML element

5.6 Debug/error logging

Functions for debugging and error messages.

Typedefs

- typedef void(*) **GGZDebugHandlerFunc** (int priority, const char *msg)

  A callback function to handle debugging output.

Enumerations

- enum **GGZCheckType** { **GGZ_CHECK_NONE** = 0x00, **GGZ_CHECK_MEM** = 0x01 }

  What memory checks should we do?

Functions

- void **ggz_debug_init** (const char **types, const char *file)

  Initialize and configure debugging for the program.

- **GGZDebugHandlerFunc ggz_debug_set_func** (**GGZDebugHandlerFunc** func)

  Set the debug handler function.

- void **ggz_debug_enable** (const char *type)

  Enable a specific type of debugging.

- void **ggz_debug_disable** (const char *type)

  Disable a specific type of debugging.

- void **ggz_debug** (const char *type, const char *fmt,...) ggz__attribute((format(printf

  Log a debugging message.

- void void **ggz_log** (const char *type, const char *fmt,...) ggz__attribute((format(printf

  Log a notice message.

- void void void **ggz_error_sys** (const char *fmt,...) ggz__attribute((format(printf

  Log a syscall error.

- void void void void **ggz_error_sys_exit** (const char *fmt,...) ggz__attribute((format(printf

  Log a fatal syscall error.

- void void void void **ggz__attribute** ((noreturn))
- void **ggz_error_msg** (const char *fmt,...) ggz__attribute((format(printf

  Log an error message.

- void void **ggz_error_msg_exit** (const char *fmt,...) ggz__attribute((format(printf

Log a fatal error message.

- void **ggz_debug_cleanup** (**GGZCheckType** check)

  Cleans up debugging state and prepares for exit.

5.6.1 Detailed Description

Functions for debugging and error messages.

5.6.2 Typedef Documentation

5.6.2.1 typedef void(*) GGZDebugHandlerFunc(int priority, const char *msg)

A callback function to handle debugging output.

A function of this type can be registered as a callback handler to handle debugging output, rather than having the output go to stderr or to a file. If this is done, each line of output will be sent directly to this function (no trailing newline will be appended).

See also:

ggz_debug_set_func (p. 38)

Parameters:

priority The priority of the log, i.e. LOG_DEBUG; see syslog()

msg The debugging output message

Note:

If your program is threaded, this function must be threadsafe.

5.6.3 Enumeration Type Documentation

5.6.3.1 enum GGZCheckType

What memory checks should we do?

See also:

ggz_debug_cleanup (p. 40)

Enumerator:

GGZ_CHECK_NONE No checks.

GGZ_CHECK_MEM Memory (leak) checks.

5.6.4 Function Documentation

5.6.4.1 void ggz_debug_init (const char ** *types*, const char * *file*)

Initialize and configure debugging for the program.

This should be called early in the program to set up the debugging routines.

Parameters:

types A null-terminated list of arbitrary string debugging "types"

file A file to write debugging output to, or NULL for none

See also:

ggz_debug (p. 39)

5.6.4.2 GGZDebugHandlerFunc ggz_debug_set_func (GGZDebugHandlerFunc *func*)

Set the debug handler function.

Call this function to register a debug handler function. NULL can be passed to disable the debug handler. If set, the debug handler function will be called to handle debugging output, overriding any file that had previously been specified.

Parameters:

func The new debug handler function

Returns:

The previous debug handler function

Note:

This function is not threadsafe (re-entrant).

5.6.4.3 void ggz_debug_enable (const char * *type*)

Enable a specific type of debugging.

Any **ggz_debug()** (p. 39) calls that use that type will then be logged.

Parameters:

type The "type" of debugging to enable

See also:

ggz_debug (p. 39)

5.6.4.4 void ggz_debug_disable (const char * *type*)

Disable a specific type of debugging.

Any **ggz_debug()** (p. 39) calls that use the given type of debugging will then not be logged.

Parameters:

type The "type" of debugging to disable

See also:

ggz_debug (p. 39)

5.6.4.5 void ggz_debug (const char * *type*, const char * *fmt*, ...)

Log a debugging message.

This function takes a debugging "type" as well as a printf-style list of arguments. It assembles the debugging message (printf-style) and logs it if the given type of debugging is enabled.

Parameters:

type The "type" of debugging (similar to a loglevel)

fmt A printf-style format string

See also:

ggz_debug_enable (p. 38), **ggz_debug_disable** (p. 38)

5.6.4.6 void void ggz_log (const char * *type*, const char * *fmt*, ...)

Log a notice message.

This function is nearly identical to **ggz_debug()** (p. 39), except that if the debugging output ends up passed to the debug handler function, the priority will be LOG_NOTICE instead of LOG_DEBUG. This is only of interest to a few programs.

Parameters:

type The "type" of debugging (similar to a loglevel)

fmt A printf-style format string

See also:

ggz_debug (p. 39), **ggz_debug_set_func** (p. 38)

5.6.4.7 void void void ggz_error_sys (const char * *fmt*, ...)

Log a syscall error.

This logs an error message in a similar manner to **ggz_debug()** (p. 39)'s debug logging. However, the logging is done regardless of whether debugging is enabled or what debugging types are set. errno and strerror are also used to create a more useful message.

Parameters:

fmt A printf-style format string

See also:

ggz_debug (p. 39)

5.6.4.8 void void void void ggz_error_sys_exit (const char * *fmt*, ...)

Log a fatal syscall error.

This logs an error message just like **ggz_error_sys()** (p. 39), and also exits the program.

Parameters:

fmt A printf-style format string

5.6.4.9 void ggz_error_msg (const char * *fmt*, ...)

Log an error message.

This logs an error message in a similar manner to **ggz_debug()** (p. 39)'s debug logging. However, the logging is done regardless of whether debugging is enabled or what debugging types are set.

Parameters:

fmt A printf-style format string

Note:

This is equivalent to ggz_debug(NULL, ...) with debugging enabled.

5.6.4.10 void void ggz_error_msg_exit (const char * *fmt*, ...)

Log a fatal error message.

This logs an error message just like **ggz_error_msg()** (p. 40), and also exits the program.

Parameters:

fmt A printf-style format string

5.6.4.11 void ggz_debug_cleanup (GGZCheckType *check*)

Cleans up debugging state and prepares for exit.

This function should be called right before the program exits. It cleans up all of the debugging state data, including writing out the memory check data (if enabled) and closing the debugging file (if enabled).

Parameters:

check A mask of things to check

5.7 Miscellaneous convenience functions

Typedefs

- typedef _GGZFile **GGZFile**

  *Structure used internally by **ggz_read_line()** (p. 43).*

Functions

- char * **ggz_xml_escape** (const char *str)

  Escape XML characters in a text string.

- char * **ggz_xml_unescape** (const char *str)

  Restore escaped XML characters into a text string.

- **GGZFile** * **ggz_get_file_struct** (int fdes)

  *Setup a file structure to use with **ggz_read_line()** (p. 43).*

- int **ggz_make_path** (const char *full)

  Create directories to fill out a path.

- char * **ggz_read_line** (**GGZFile** *file)

  Read a line of arbitrary length from a file.

- void **ggz_free_file_struct** (**GGZFile** *file)

  *Deallocate a file structure allocated via **ggz_get_file_struct()** (p. 42).*

- int **ggz_strcmp** (const char *s1, const char *s2)

  String comparison function that is safe with NULLs.

- int **ggz_strcasecmp** (const char *s1, const char *s2)

  Case-insensitive string comparison function that is safe with NULLs The function returns an integer less than, equal to, or greater than zero if s1 is found, respectively, to be less than, to match, or be greater than s2.

5.7.1 Function Documentation

5.7.1.1 char* ggz_xml_escape (const char * *str*)

Escape XML characters in a text string.

Parameters:

str The string to encode

Returns:

A pointer to a dynamically allocated string with XML characters replaced with ampersand tags, or NULL on error

Note:

The dynamic memory is allocated using **ggz_malloc()** (p. 10) and the caller is expected to later free this memory using **ggz_free()** (p. 11). If the original string did not contain any characters which required escaping a **ggz_strdup()** (p. 11) copy is returned.

5.7.1.2 char* ggz_xml_unescape (const char * *str*)

Restore escaped XML characters into a text string.

Parameters:

str The string to decode

Returns:

A pointer to a dynamically allocated string with XML ampersand tags replaced with their normal ASCII characters, or NULL on error

Note:

The dyanmic memory is allocated using **ggz_malloc()** (p. 10) and the caller is expected to later free this memory using **ggz_free()** (p. 11). If the original string did not contain any characters which required decoding, a **ggz_strdup()** (p. 11) copy is returned.
When using expat, incoming text is automatically unescaped by the expat library. It is therefore generally not necessary to use this function with expat.

5.7.1.3 GGZFile* ggz_get_file_struct (int *fdes*)

Setup a file structure to use with **ggz_read_line()** (p. 43).

Parameters:

fdes A preopened integer file descriptor to read from

Returns:

A pointer to a dynamically allocated **GGZFile** (p. 75) structure

Note:

The user MUST have opened the requested file for reading before using this function. When finished using **ggz_read_line()** (p. 43), the user should cleanup this struct using **ggz_free_file_struct()** (p. 43).

5.7.1.4 int ggz_make_path (const char * *full*)

Create directories to fill out a path.

New directories are created with permissions 700 (S_IRWXU).

Parameters:

full The full path to be created.

Returns:

0 on success; -1 on failure

5.7.1.5 char* ggz_read_line (GGZFile * *file*)

Read a line of arbitrary length from a file.

Parameters:

file A **GGZFile** (p. 75) structure allocated via **ggz_get_file_struct()** (p. 42)

Returns:

A NULL terminated line from the file of arbitrary length or NULL at end of file

Note:

The dynamic memory is allocated using **ggz_malloc()** (p. 10) and the caller is expected to later free this memory using **ggz_free()** (p. 11).

5.7.1.6 void ggz_free_file_struct (GGZFile * *file*)

Deallocate a file structure allocated via **ggz_get_file_struct()** (p. 42).

Parameters:

file A **GGZFile** (p. 75) structure allocated via **ggz_get_file_struct()** (p. 42)

Note:

The caller is expected to close the I/O file before or after freeing the file structure.

5.7.1.7 int ggz_strcmp (const char * *s1*, const char * *s2*)

String comparison function that is safe with NULLs.

Parameters:

s1 First string to compare

s2 Second string to compare

Returns:

An integer less than, equal to, or greater than zero if s1 is found, respectively, to be less than, to match, or be greater than s2

Note:

NULL in considered to be less than any non-NULL string and equal to itself

5.7.1.8 int ggz_strcasecmp (const char * *s1*, const char * *s2*)

Case-insensitive string comparison function that is safe with NULLs The function returns an integer less than, equal to, or greater than zero if s1 is found, respectively, to be less than, to match, or be greater than s2.

NULL in considered to be less than any non-NULL string and equal to itself

Parameters:

s1 First string to compare

s2 Second string to compare

Returns:

The comparison value.

5.8 Easysock IO

Simple functions for reading/writing binary data across file descriptors.

Defines

- #define **GGZ_SOCKET_DEBUG** "socket"

  ggz_debug debugging type for Easysock debugging.

Typedefs

- typedef void(*) **ggzIOError** (const char *msg, const **GGZIOType** type, const int fd, const **GGZDataType** data)

  An error function type.

- typedef void(*) **ggzIOExit** (int status)

  An exit function type.

- typedef void(*) **ggzNetworkNotify** (const char *address, int socket)

  A network resolver function type.

Enumerations

- enum **GGZIOType** { **GGZ_IO_CREATE**, **GGZ_IO_READ**, **GGZ_IO_WRITE**, **GGZ_IO_ALLOCATE** }

  An error type for the GGZ socket functions.

- enum **GGZDataType** {
  GGZ_DATA_NONE, **GGZ_DATA_CHAR**, **GGZ_DATA_INT**, **GGZ_DATA_STRING**,
  GGZ_DATA_FD }

  A data type for the GGZ socket function error handler.

- enum **GGZSockType** { **GGZ_SOCK_SERVER**, **GGZ_SOCK_CLIENT** }

  A socket type.

Functions

- int **ggz_set_io_error_func** (**ggzIOError** func)

  Set the ggz/easysock error handling function.

- **ggzIOError ggz_remove_io_error_func** (void)

  Remove the ggz/easysock error handling function.

- int **ggz_set_io_exit_func** (**ggzIOExit** func)

Set the ggz/easysock exit function.

- **ggzIOExit ggz_remove_io_exit_func** (void)
  Remove the ggz/easysock exit function.
- unsigned int **ggz_get_io_alloc_limit** (void)
  Get libggz's limit on memory allocation.
- unsigned int **ggz_set_io_alloc_limit** (const unsigned int limit)
  Set libggz's limit on memory allocation.
- int **ggz_init_network** (void)
  Initialize the network.
- int **ggz_set_network_notify_func** (**ggzNetworkNotify** func)
  Set the ggz/easysock resolver notification function.
- const char * **ggz_resolvename** (const char *name)
  Resolve a host name.
- const char * **ggz_getpeername** (int fd, int resolve)
  Get the IP address or host name of a connected peer.
- int **ggz_make_socket** (const **GGZSockType** type, const unsigned short port, const char *server)
  Make a socket connection.
- int **ggz_make_socket_or_die** (const **GGZSockType** type, const unsigned short port, const char *server)
  Make a socket connection, exiting on error.
- int **ggz_make_unix_socket** (const **GGZSockType** type, const char *name)
  Connect to a unix domain socket.
- int **ggz_make_unix_socket_or_die** (const **GGZSockType** type, const char *name)
  Connect to a unix domain socket, exiting on error.
- int **ggz_write_char** (const int sock, const char data)
  Write a character value to the given socket.
- void **ggz_write_char_or_die** (const int sock, const char data)
  Write a character value to the given socket, exiting on error.
- int **ggz_read_char** (const int sock, char *data)
  Read a character value from the given socket.
- void **ggz_read_char_or_die** (const int sock, char *data)
  Read a character value from the given socket, exiting on error.
- int **ggz_write_int** (const int sock, const int data)

Write an integer to the socket in network byte order.

- void **ggz_write_int_or_die** (const int sock, const int data)

  Write an integer to the socket, exiting on error.

- int **ggz_read_int** (const int sock, int *data)

  Read an integer from the socket in network byte order.

- void **ggz_read_int_or_die** (const int sock, int *data)

  Read an integer from the socket, exiting on error.

- int **ggz_write_string** (const int sock, const char *data)

  Write a string to the given socket.

- void **ggz_write_string_or_die** (const int sock, const char *data)

  Write a string to the given socket, exiting on error.

- int **ggz_va_write_string** (const int sock, const char *fmt,...) ggz__attribute((format(printf

  Write a printf-style formatted string to the given socket.

- int void **ggz_va_write_string_or_die** (const int sock, const char *fmt,...) ggz__attribute((format(printf

  Write a formatted string to the socket, exiting on error.

- int void int **ggz_read_string** (const int sock, char *data, const unsigned int len)

  Read a string from the given socket.

- void **ggz_read_string_or_die** (const int sock, char *data, const unsigned int len)

  Read a string from the given socket, exiting on error.

- int **ggz_read_string_alloc** (const int sock, char **data)

  Read and allocate a string from the given socket.

- void **ggz_read_string_alloc_or_die** (const int sock, char **data)

  Read and allocate string from the given socket, exiting on error.

- int **ggz_write_fd** (const int sock, int sendfd)

  Write a file descriptor to the given (local) socket.

- int **ggz_read_fd** (const int sock, int *recvfd)

  Read a file descriptor from the given (local) socket.

- int **ggz_writen** (const int sock, const void *vdata, size_t n)

  Write a sequence of bytes to the socket.

- int **ggz_readn** (const int sock, void *data, size_t n)

  Read a sequence of bytes from the socket.

5.8.1 Detailed Description

Simple functions for reading/writing binary data across file descriptors.

5.8.2 Define Documentation

5.8.2.1 #define GGZ_SOCKET_DEBUG "socket"

ggz_debug debugging type for Easysock debugging.

To enable socket debugging, add this to the list of debugging types.

See also:

ggz_debug_enable (p. 38)

5.8.3 Typedef Documentation

5.8.3.1 typedef void(*) ggzIOError(const char *msg, const GGZIOType type, const int fd, const GGZDataType data)

An error function type.

This function type will be called when there is an error in a GGZ socket function.

Parameters:

msg The strerror message associated with the error

type The type of error that occurred

fd The socket on which the error occurred, or -1 if not applicable

data Extra data associated with the error

5.8.3.2 typedef void(*) ggzIOExit(int status)

An exit function type.

This function type will be called to exit the program.

Parameters:

status The exit value

5.8.3.3 typedef void(*) ggzNetworkNotify(const char *address, int socket)

A network resolver function type.

This function type will be called whenever a hostname has been resolved or a socket has been created asynchronously.

Parameters:

status The IP address of the host name

socket File descriptor or error code like in ggz_make_socket

5.8.4 Enumeration Type Documentation

5.8.4.1 enum GGZIOType

An error type for the GGZ socket functions.

If there is a GGZ socket error, the registered error handler will be called and will be given one of these error types.

Enumerator:

GGZ_IO_CREATE Error creating a socket.
GGZ_IO_READ Error reading from a socket.
GGZ_IO_WRITE Error writing to a socket.
GGZ_IO_ALLOCATE Error when the allocation limit is exceeded.

5.8.4.2 enum GGZDataType

A data type for the GGZ socket function error handler.

If there is a GGZ socket error, the registered error handler will be called and will be given one of these error data types.

Enumerator:

GGZ_DATA_NONE No data is associated with the error.
GGZ_DATA_CHAR The error occurred while dealing with a char.
GGZ_DATA_INT The error occurred in dealing with an integer.
GGZ_DATA_STRING The error occurred in dealing with a string.
GGZ_DATA_FD Error while dealing with a file descriptor.

5.8.4.3 enum GGZSockType

A socket type.

These socket types are used by **ggz_make_socket()** (p. 53) and friends to decide what actions are necessary in making a connection.

Enumerator:

GGZ_SOCK_SERVER Just listen on a particular port.
GGZ_SOCK_CLIENT Connect to a particular port of a server.

5.8.5 Function Documentation

5.8.5.1 int ggz_set_io_error_func (ggzIOError *func*)

Set the ggz/easysock error handling function.

Any time an error occurs in a GGZ socket function, the registered error handling function will be called. Use this function to register a new error function. Any previous error function will be discarded.

Parameters:

func The new error-handling function

Returns:

0

Todo

Shouldn't this function return a void or ggzIOError?

5.8.5.2 ggzIOError ggz_remove_io_error_func (void)

Remove the ggz/easysock error handling function.

The default behavior when a socket failure occurs in one of the GGZ socket functions is to do nothing (outside of the function's return value). This may be overridden by registering an error handler with **ggz_set_io_error_func()** (p. 49), but the behavior may be returned by calling this function to remove the error handler.

Returns:

The previous error-handling function, or NULL if none.

5.8.5.3 int ggz_set_io_exit_func (ggzIOExit *func*)

Set the ggz/easysock exit function.

Any of the *_or_die() functions will call the set exit function if there is an error. If there is no set function, exit() will be called.

Parameters:

func The newly set exit function

Returns:

0

Todo

Shouldn't this return a void?

5.8.5.4 ggzIOExit ggz_remove_io_exit_func (void)

Remove the ggz/easysock exit function.

This removes the existing exit function, if one is set. exit() will then be called to exit the program.

Returns:

The old exit function (or NULL if none)

5.8.5.5 unsigned int ggz_get_io_alloc_limit (void)

Get libggz's limit on memory allocation.

Returns:

The limit to allow on ggz_read_*_alloc() calls, in bytes

See also:

ggz_set_io_alloc_limit (p. 51)

5.8.5.6 unsigned int ggz_set_io_alloc_limit (const unsigned int *limit*)

Set libggz's limit on memory allocation.

In functions of the form ggz_read_*_alloc(), libggz will itself allocate memory for the * object that is being read in. This presents an obvious security concern, so we limit the amount of memory that can be allocated. The default value is 32,767 bytes, but it can be changed by calling this function.

Parameters:

limit The new limit to allow on alloc-style calls, in bytes

Returns:

The previous limit

See also:

ggz_get_io_alloc_limit (p. 51)
ggz_read_string_alloc (p. 57)

5.8.5.7 int ggz_init_network (void)

Initialize the network.

This function will do anything necessary to initialize the network to use sockets (this is needed on some platforms). It should be called at least once before any sockets are put to use. Calling it more than once is harmless. It will be called automatically at the start of all GGZ socket creation functions.

Returns:

0 on success, negative on failure

5.8.5.8 int ggz_set_network_notify_func (ggzNetworkNotify *func*)

Set the ggz/easysock resolver notification function.

This function will be called whenever a resolving task submitted to **ggz_resolvename()** (p. 52) or **ggz_make_socket()** (p. 53) has finished.

Parameters:

func The newly set resolver notification function

Returns:

0

Todo

Shouldn't this return a void? (from ggz_set_io_exit_func)

5.8.5.9 const char* ggz_resolvename (const char * *name*)

Resolve a host name.

In order to prevent blocking GUIs, this function can handle resolving a hostname into a numerical address asynchronously. The notification function will be called whenever it finishes. It receives as its argument the address, which might still be the same hostname in the case of errors. The result should be passed to gethostbyname() to receive the network data structures. If no notification function is set, this function returns the hostname as it is, without any lookup. If no GAI support is available, but a notification function is set, it is called with the unresolved hostname, too.

Parameters:

name Hostname to resolve

Returns:

The hostname in case no notification function is set, or NULL

Todo

Should this resolve synchronously in the special cases above?

5.8.5.10 const char* ggz_getpeername (int *fd*, int *resolve*)

Get the IP address or host name of a connected peer.

This function tells about the IP address of the peer which is connected to the specified socket. If resolving is enabled, then the hostname is returned instead. In this case, resolving will be a blocking operation.

Parameters:

fd Local end file descriptor of the connection

resolve Whether or not to resolve the host name

Returns:

IP address or host name of peer, or NULL on error

Note:

The string must be **ggz_free()** (p. 11)d afterwards

5.8.5.11 int ggz_make_socket (const GGZSockType *type*, const unsigned short *port*, const char * *server*)

Make a socket connection.

This function makes a TCP socket connection.

- For a server socket, we'll just listen on the given port and accept the first connection that is made there.
- For a client socket, we'll connect to a server that is (hopefully) listening at the given port and hostname.

Note that when a ggzNetworkNotify callback has been set up, this function returns immediately and creates the socket later on.

Parameters:

type The type of socket (server or client)

port The port to listen/connect to

server The server hostname for clients, the interface address else

Returns:

File descriptor on success, -1 on creation error, -2 on lookup error, -3 when using asynchronous creation

5.8.5.12 int ggz_make_socket_or_die (const GGZSockType *type*, const unsigned short *port*, const char * *server*)

Make a socket connection, exiting on error.

Aside from the error condition, this is identical to **ggz_make_socket()** (p. 53).

5.8.5.13 int ggz_make_unix_socket (const GGZSockType *type*, const char * *name*)

Connect to a unix domain socket.

This function connects to a unix domain socket, a socket associated with a file on the VFS.

- For a server socket, we unlink the socket file first and then create it.
- For a client socket, we connect to a pre-existing socket file.

Parameters:

type The type of socket (server or client)

name The name of the socket file

Returns:

The socket FD on success, -1 on error

Note:

When possible, this should not be used. Use socketpair() instead.

5.8.5.14 int ggz_make_unix_socket_or_die (const GGZSockType *type*, const char * *name*)

Connect to a unix domain socket, exiting on error.

Aside from the error condition, this is identical to **ggz_make_unix_socket()** (p. 53).

5.8.5.15 int ggz_write_char (const int *sock*, const char *data*)

Write a character value to the given socket.

This function will write a single character to the socket. The character will be readable at the other end with ggz_read_char.

Parameters:

sock The socket file descriptor to write to

data A single character to write

Returns:

0 on success, -1 on error

5.8.5.16 void ggz_write_char_or_die (const int *sock*, const char *data*)

Write a character value to the given socket, exiting on error.

Parameters:

sock The socket file descriptor to write to

data A single character to write

Note:

Aside from error handling, this is identical to **ggz_write_char()** (p. 54).

5.8.5.17 int ggz_read_char (const int *sock*, char * *data*)

Read a character value from the given socket.

This function will read a single character (as written by **ggz_write_char()** (p. 54)) from a socket. It places the value into the character pointed to.

Parameters:

sock The socket file descriptor to read from

data A pointer to a single character

Returns:

0 on success, -1 on error

5.8.5.18 void ggz_read_char_or_die (const int *sock*, char * *data*)

Read a character value from the given socket, exiting on error.

Parameters:

sock The socket file descriptor to read from

data A pointer to a single character

Note:

Aside from error handling, this is identical to **ggz_read_char()** (p. 54).

5.8.5.19 int ggz_write_int (const int *sock*, const int *data*)

Write an integer to the socket in network byte order.

ggz_write_int() (p. 55) and **ggz_read_int()** (p. 55) can be used to send an integer across a socket. The integer will be sent in network byte order, so the functions may safely be used across a network. Note, though, that it is probably not safe to use this function to send structs or other data that may use a different representation than a simple integer.

Parameters:

sock The socket to write to

data The integer to write

Returns:

0 on success, -1 on error

5.8.5.20 void ggz_write_int_or_die (const int *sock*, const int *data*)

Write an integer to the socket, exiting on error.

Aside from the error condition, this function is identical to **ggz_write_int()** (p. 55).

5.8.5.21 int ggz_read_int (const int *sock*, int * *data*)

Read an integer from the socket in network byte order.

See also:

ggz_write_int (p. 55)

Parameters:

sock The socket to read from

data A pointer to an integer in which to place the data

Returns:

0 on success, -1 on error

5.8.5.22 void ggz_read_int_or_die (const int *sock*, int * *data*)

Read an integer from the socket, exiting on error.

Aside from the error condition, this function is identical to ggz_read_int.

5.8.5.23 int ggz_write_string (const int *sock*, const char * *data*)

Write a string to the given socket.

This function will write a full string to the given socket. The string may be read at the other end by **ggz_read_string()** (p. 57) and friends.

Parameters:

sock The socket file descriptor to write to

data A pointer to the string to write

Returns:

0 on success, -1 on error

5.8.5.24 void ggz_write_string_or_die (const int *sock*, const char * *data*)

Write a string to the given socket, exiting on error.

Aside from the error condition, this function is identical to **ggz_write_string()** (p. 56).

5.8.5.25 int ggz_va_write_string (const int *sock*, const char * *fmt*, ...)

Write a printf-style formatted string to the given socket.

This function allows a format string and a list of arguments to be passed in. The function will assemble the string (printf-style) and write it to the socket.

Parameters:

sock The socket file descriptor to write to

fmt A printf-style formatting string

... A printf-style list of arguments

Note:

This function will write identically to **ggz_write_string()** (p. 56).

5.8.5.26 int void ggz_va_write_string_or_die (const int *sock*, const char * *fmt*, ...)

Write a formatted string to the socket, exiting on error.

Aside from the error condition, this function is identical to ggz_va_write_string.

5.8.5.27 int void int ggz_read_string (const int *sock*, char * *data*, const unsigned int *len*)

Read a string from the given socket.

This function will read a full string from the given socket. The string may be written at the other end by **ggz_write_string()** (p. 56) and friends. The length of the string is given as well to avoid buffer overruns; any characters beyond this will be lost.

Parameters:

sock The socket file descriptor to read from

data A pointer to the string to read; it will be changed

len The length of the string pointed to by data

Returns:

0 on success, -1 on error

5.8.5.28 void ggz_read_string_or_die (const int *sock*, char * *data*, const unsigned int *len*)

Read a string from the given socket, exiting on error.

Aside from the error condition, this function is identical to **ggz_read_string()** (p. 57).

5.8.5.29 int ggz_read_string_alloc (const int *sock*, char ** *data*)

Read and allocate a string from the given socket.

This function reads a string from the socket, just like **ggz_read_string()** (p. 57). But instead of passing in a pre-allocated buffer to write in, here we pass a pointer to a string pointer:

```
char* str;
if (ggz_read_string_alloc(fd, &str) >= 0) {
    // ... handle the string ...
    ggz_free(str);
}
```

Parameters:

sock The socket file descriptor to read from

data A pointer to an empty string pointer

Returns:

0 on success, -1 on error

Note:

The use of this function is a security risk.

See also:

ggz_set_io_alloc_limit (p. 51)

5.8.5.30 void ggz_read_string_alloc_or_die (const int *sock*, char ** *data*)

Read and allocate string from the given socket, exiting on error.

Aside from the error condition, this function is identical to **ggz_read_string_alloc()** (p. 57).

5.8.5.31 int ggz_write_fd (const int *sock*, int *sendfd*)

Write a file descriptor to the given (local) socket.

ggz_write_fd() (p. 58) and **ggz_read_fd()** (p. 58) handle the rather tricky task of sending a file descriptor across a socket. The FD is written with **ggz_write_fd()** (p. 58) and can be read at the other end by **ggz_read_fd()** (p. 58). Note that this will only work for local sockets (i.e. not over the network). Many thanks to Richard Stevens and his wonderful books, from which these functions come.

Parameters:

sock The socket to write to

sendfd The FD to send across the socket

Returns:

0 on success, -1 on error

5.8.5.32 int ggz_read_fd (const int *sock*, int * *recvfd*)

Read a file descriptor from the given (local) socket.

See also:

ggz_write_fd (p. 58)

Parameters:

sock The socket to read from

recvfd The FD that is read

Returns:

0 on success, -1 on error

5.8.5.33 int ggz_writen (const int *sock*, const void * *vdata*, size_t *n*)

Write a sequence of bytes to the socket.

ggz_writen() (p. 58) and **ggz_readn()** (p. 59) are used to send an arbitrary quantity of raw data across the a socket. The data is written with **ggz_writen()** (p. 58) and can be read at the other end with **ggz_readn()** (p. 59). Many thanks to Richard Stevens and his wonderful books, from which these functions come.

Parameters:

sock The socket to write to

vdata A pointer to the data to write

n The number of bytes of data to write from vdata

Returns:

0 on success, -1 on error

5.8.5.34 int ggz_readn (const int *sock*, void * *data*, size_t *n*)

Read a sequence of bytes from the socket.

See also:

ggz_writen (p. 58)

Parameters:

sock The socket to read from

data A pointer a buffer of size >= n in which to place the data

n The number of bytes to read

Returns:

0 on success, -1 on error

Note:

You must know how much data you want BEFORE calling this function.

5.9 Security functions

All functions related to encryption and encoding go here.

Data Structures

- struct **hash_t**
  Hash data structure.

Enumerations

- enum **GGZTLSType** { **GGZ_TLS_CLIENT**, **GGZ_TLS_SERVER** }
  TLS operation mode.
- enum **GGZTLSVerificationType** { **GGZ_TLS_VERIFY_NONE**, **GGZ_TLS_VERIFY_PEER** }
  TLS verification type.

Functions

- **hash_t ggz_hash_create** (const char *algo, const char *text)
  Create a hash over a text.
- **hash_t ggz_hmac_create** (const char *algo, const char *text, const char *secret)
  Create a HMAC hash over a text.
- char * **ggz_base16_encode** (const char *text, int length)
  Encodes text to base16.
- char * **ggz_base64_encode** (const char *text, int length)
  Encodes text to base64.
- char * **ggz_base64_decode** (const char *text, int length)
  Decodes text from base64.
- void **ggz_tls_init** (const char *certfile, const char *keyfile, const char *password)
  Initialize TLS support on the server side.
- int **ggz_tls_support_query** (void)
  Check TLS support.
- const char * **ggz_tls_support_name** (void)
  Name of the TLS implementation.
- int **ggz_tls_enable_fd** (int fdes, **GGZTLSType** whoami, **GGZTLSVerificationType** verify)
  Enable TLS for a file descriptor.

- int **ggz_tls_disable_fd** (int fdes)

  Disable TLS for a file descriptor.

- size_t **ggz_tls_write** (int fd, void *ptr, size_t n)

  Write some bytes to a secured file descriptor.

- size_t **ggz_tls_read** (int fd, void *ptr, size_t n)

  Read from a secured file descriptor.

5.9.1 Detailed Description

All functions related to encryption and encoding go here.

Encryption functions use gcrypt, and will always fail if support for gcrypt has not been compiled in. Encoding functions will always be available.

5.9.2 Enumeration Type Documentation

5.9.2.1 enum GGZTLSType

TLS operation mode.

Hints whether the TLS handshake will happen in either client or server mode.

See also:

ggz_tls_enable_fd (p. 64)

Enumerator:

GGZ_TLS_CLIENT Operate as client.

GGZ_TLS_SERVER Operate as server.

5.9.2.2 enum GGZTLSVerificationType

TLS verification type.

The authentication (verification) model to be used for the handshake. None means that no certificate is validated.

See also:

ggz_tls_enable_fd (p. 64)

Enumerator:

GGZ_TLS_VERIFY_NONE Don't perform verification.

GGZ_TLS_VERIFY_PEER Perform validation of the server's cert.

5.9.3 Function Documentation

5.9.3.1 hash_t ggz_hash_create (const char * *algo*, const char * *text*)

Create a hash over a text.

A hash sum over a given text is created, using the given algorithm. Space is allocated as needed.

Parameters:

algo The algorithm, like md5 or sha1

text Plain text used to calculate the hash sum

Returns:

Hash value in a structure

5.9.3.2 hash_t ggz_hmac_create (const char * *algo*, const char * *text*, const char * *secret*)

Create a HMAC hash over a text.

Creates a hash sum using a secret key. Space is allocated as needed and must be freed afterwards.

Parameters:

algo The algorithm to use, like md5 or sha1

text Plain text used to calculate the hash sum

secret Secret key to be used for the HMAC creation

Returns:

Hash value in a structure

5.9.3.3 char* ggz_base16_encode (const char * *text*, int *length*)

Encodes text to base16.

Plain text with possibly unsafe characters is converted to the base16 (hex) format through this function. The returned string is allocated internally and must be freed.

Parameters:

text Plain text to encode

length Length of the text (which may contain binary characters), in bytes

Returns:

Base16 representation of the text

5.9.3.4 char* ggz_base64_encode (const char * *text*, int *length*)

Encodes text to base64.

Plain text with possibly unsafe characters is converted to the base64 format through this function. The returned string is allocated internally and must be freed.

Parameters:

text Plain text to encode

length Length of the text (which may contain binary characters), in bytes

Returns:

Base64 representation of the text

5.9.3.5 char* ggz_base64_decode (const char * *text*, int *length*)

Decodes text from base64.

This is the reverse function to **ggz_base64_encode()** (p. 63). It will also allocate space as needed.

Parameters:

text Text in base64 format

length Length of the text, in bytes

Returns:

Native representation, may contain binary characters

5.9.3.6 void ggz_tls_init (const char * *certfile*, const char * *keyfile*, const char * *password*)

Initialize TLS support on the server side.

This function sets up the necessary initialization values. It must be called by both the client and the server before any other TLS operations can take place. The client can pass NULL values for all parameters.

Parameters:

certfile File containing the certificate, or NULL

keyfile File containing the private key, or NULL

password Password to the private key, or NULL

5.9.3.7 int ggz_tls_support_query (void)

Check TLS support.

Checks if real TLS support is available or communication will fall back to unencrypted connections. Even in the case of support, individual connections might still be unencrypted if the handshake fails.

Returns:

1 if TLS is supported, 0 if no support is present

See also:

ggz_tls_enable_fd (p. 64)

5.9.3.8 const char* ggz_tls_support_name (void)

Name of the TLS implementation.

Returns the name of the TLS layer implementation used to encrypt connections.

Returns:

TLS implementation name, or NULL if no TLS support is present

See also:

ggz_tls_support_query (p. 63)

5.9.3.9 int ggz_tls_enable_fd (int *fdes*, GGZTLSType *whoami*, GGZTLSVerificationType *verify*)

Enable TLS for a file descriptor.

A TLS handshake is performed for an existing connection on the given file descriptor. On success, all consecutive data will be encrypted.

Parameters:

fdes File descriptor in question

whoami Operation mode (client or server)

verify Verification mode

Returns:

1 on success, 0 on failure

5.9.3.10 int ggz_tls_disable_fd (int *fdes*)

Disable TLS for a file descriptor.

An existing TLS connection is reset to a normal connection on which all communication happens without encryption.

Parameters:

fdes File descriptor in question

Returns:

1 on success, 0 on failure

5.9.3.11 size_t ggz_tls_write (int *fd*, void * *ptr*, size_t *n*)

Write some bytes to a secured file descriptor.

This function acts as a TLS-aware wrapper for write(2).

Parameters:

fd File descriptor to use

ptr Pointer to the data to write

n Length of the data to write, in bytes

Returns:

Actual number of bytes written

5.9.3.12 size_t ggz_tls_read (int *fd*, void * *ptr*, size_t *n*)

Read from a secured file descriptor.

This function acts as a TLS-aware wrapper for read(2).

Parameters:

fd File descriptor to use

ptr Pointer to a buffer to store the data into

n Number of bytes to read, and minimum size of the buffer

Returns:

Actually read number of bytes

Chapter 6

LibGGZ Data Structure Documentation

6.1 _GGZList Struct Reference

Simple doubly-linked list.

`#include <ggz.h>`

Data Fields

- **GGZListEntry * head**

  Pointer to the first node in the list.

- **GGZListEntry * tail**

  Pointer to the last node in the list.

- **ggzEntryCompare compare_func**

  Function used to compare data entries.

- **ggzEntryCreate create_func**

  Function used to copy data entries.

- **ggzEntryDestroy destroy_func**

  Function used to destroy data entries.

- int **options**

  List options.

- int **entries**

  The current number of list entries (ie.

6.1.1 Detailed Description

Simple doubly-linked list.

Do not access these members directly. Instead use accessor functions.

6.1.2 Field Documentation

6.1.2.1 int _GGZList::entries

The current number of list entries (ie.

list length)

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

- **ggz.h**

6.2 _GGZListEntry Struct Reference

A single entry in a **GGZList** (p. 24).

`#include <ggz.h>`

Data Fields

- void * **data**

  Pointer to data for this node.

- **_GGZListEntry** * **next**

  Pointer to next nodes in the list.

- **_GGZListEntry** * **prev**

  Pointer to previous node in the list.

6.2.1 Detailed Description

A single entry in a **GGZList** (p. 24).

Do not access these members directly, but use the **ggz_list_get_data()** (p. 27), **ggz_list_next()** (p. 26) and **ggz_list_prev()** (p. 26) accessor functions instead.

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

- **ggz.h**

6.3 _GGZXMLElement Struct Reference

Object representing a single XML element.

`#include <ggz.h>`

Data Fields

- char * **tag**

  The name of the element.

- char * **text**

  Text content of an element.

- **GGZList** * **attributes**

  List of attributes on the element.

- void * **data**

  Extra data associated with tag (usually gleaned from children).

- void(* **free**)(struct **_GGZXMLElement** *)

  Function to free allocated memory.

- void(* **process**)(void *, struct **_GGZXMLElement** *)

  Function to "process" tag.

6.3.1 Detailed Description

Object representing a single XML element.

Except for "process", do not access these members directly. Instead use the provided accessor functions. "process" is meant to be invoked as a method on instances of GGZXMLElement.

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

- **ggz.h**

6.4 hash_t Struct Reference

Hash data structure.

`#include <ggz.h>`

Data Fields

- char * **hash**
  Hash value.
- int **hashlen**
  Length of the hash value, in bytes.

6.4.1 Detailed Description

Hash data structure.

Contains a string and its length, so that NULL-safe functions are possible.

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

- **ggz.h**

Chapter 7

LibGGZ File Documentation

7.1 ggz.h File Reference

Author:

Brent M.

```
#include <sys/types.h>
```

Data Structures

- struct **_GGZListEntry**
  *A single entry in a **GGZList** (p. 24).*
- struct **_GGZList**
  Simple doubly-linked list.
- struct **_GGZXMLElement**
  Object representing a single XML element.
- struct **hash_t**
  Hash data structure.

Defines

- #define **LIBGGZ_VERSION_MAJOR** 0
- #define **LIBGGZ_VERSION_MINOR** 0
- #define **LIBGGZ_VERSION_MICRO** 14
- #define **LIBGGZ_VERSION_IFACE** "5:0:3"
- #define **ggz__attribute**(att)
  Allow easy use of GCC's "attribute" macro for debugging.
- #define **_GGZFUNCTION_** __FUNCTION__
- #define **GGZ_MEM_DEBUG** "ggz_mem"

Debugging type for memory debugging.

- #define **ggz_malloc**(size) _ggz_malloc(size, _GGZFUNCTION_ " in " __FILE__, __LINE__)

  Macro for memory allocation.

- #define **ggz_realloc**(mem, size) _ggz_realloc(mem, size, _GGZFUNCTION_ " in " __FILE__, __LINE__)

  Macro for resizing previously allocated memory.

- #define **ggz_free**(mem) _ggz_free(mem, _GGZFUNCTION_ " in " __FILE__, __LINE__)

  Macro for freeing memory previously allocated.

- #define **ggz_strdup**(string) _ggz_strdup(string, _GGZFUNCTION_ " in " __FILE__, __LINE__)

  Macro for duplicating string.

- #define **GGZ_CONF_DEBUG** "ggz_conf"

  Debugging type for config-file debugging.

- #define **GGZ_LIST_REPLACE_DUPS** 0x00

  Overwrite duplicate values on insert.

- #define **GGZ_LIST_ALLOW_DUPS** 0x01

  Allow duplicate data entries to exist in the list.

- #define **GGZ_SOCKET_DEBUG** "socket"

  ggz_debug debugging type for Easysock debugging.

Typedefs

- typedef int(*) **ggzEntryCompare** (const void *a, const void *b)

  *A function type for doing data comparison on two items in a **GGZList** (p. 24).*

- typedef void *(*) **ggzEntryCreate** (void *data)

  *A function type for creating a copy of a data item for insertion into a **GGZList** (p. 24).*

- typedef void(*) **ggzEntryDestroy** (void *data)

  *A function type to destroy an entry in a **GGZList** (p. 24).*

- typedef _**GGZListEntry GGZListEntry**

  *A single entry in a **GGZList** (p. 24).*

- typedef _**GGZList GGZList**

  Simple doubly-linked list.

- typedef _**GGZList GGZStack**

  *Simple implementation of stacks using **GGZList** (p. 24).*

- typedef _GGZXMLElement **GGZXMLElement**

  Object representing a single XML element.

- typedef void(*) **GGZDebugHandlerFunc** (int priority, const char *msg)

  A callback function to handle debugging output.

- typedef _GGZFile **GGZFile**

  Structure used internally by ***ggz_read_line()*** *(p. 43).*

- typedef void(*) **ggzIOError** (const char *msg, const **GGZIOType** type, const int fd, const **GGZDataType** data)

  An error function type.

- typedef void(*) **ggzIOExit** (int status)

  An exit function type.

- typedef void(*) **ggzNetworkNotify** (const char *address, int socket)

  A network resolver function type.

Enumerations

- enum **GGZConfType** { **GGZ_CONF_RDONLY** = ((unsigned char) 0x01), **GGZ_CONF_RDWR** = ((unsigned char) 0x02), **GGZ_CONF_CREATE** = ((unsigned char) 0x04) }

  Specifies the mode for opening a configuration file.

- enum **GGZCheckType** { **GGZ_CHECK_NONE** = 0x00, **GGZ_CHECK_MEM** = 0x01 }

  What memory checks should we do?

- enum **GGZIOType** { **GGZ_IO_CREATE**, **GGZ_IO_READ**, **GGZ_IO_WRITE**, **GGZ_IO_ALLOCATE** }

  An error type for the GGZ socket functions.

- enum **GGZDataType** {

  GGZ_DATA_NONE, **GGZ_DATA_CHAR**, **GGZ_DATA_INT**, **GGZ_DATA_STRING**,

  GGZ_DATA_FD }

  A data type for the GGZ socket function error handler.

- enum **GGZSockType** { **GGZ_SOCK_SERVER**, **GGZ_SOCK_CLIENT** }

  A socket type.

- enum **GGZTLSType** { **GGZ_TLS_CLIENT**, **GGZ_TLS_SERVER** }

  TLS operation mode.

- enum **GGZTLSVerificationType** { **GGZ_TLS_VERIFY_NONE**, **GGZ_TLS_VERIFY_PEER** }

TLS verification type.

Functions

- void * **_ggz_malloc** (const size_t size, const char *tag, int line)

  Function to actually perform memory allocation.

- void * **_ggz_realloc** (const void *ptr, const size_t size, const char *tag, int line) ggz__attribute((warn_unused_result))

  Function to perform memory reallocation.

- int **_ggz_free** (const void *ptr, const char *tag, int line)

  Function to free allocated memory.

- char * **_ggz_strdup** (const char *ptr, const char *tag, int line) ggz__attribute((warn_unused_result))

  Function to copy a string.

- char * **ggz_strncpy** (char *dst, const char *src, size_t n)

  Safe version of strncpy.

- int **ggz_memory_check** (void)

  Check memory allocated against memory freed and display any discrepancies.

- void **ggz_conf_cleanup** (void)

  Closes all open configuration files.

- void **ggz_conf_close** (int handle)

  Closes one configuration file.

- int **ggz_conf_parse** (const char *path, const **GGZConfType** options)

  Opens a configuration file and parses the variables so they can be retrieved with the access functions.

- int **ggz_conf_commit** (int handle)

  Commits any changed variables to the configuration file.

- int **ggz_conf_write_string** (int handle, const char *section, const char *key, const char *value)

  Writes a string value to a section and key in an open configuration file.

- int **ggz_conf_write_int** (int handle, const char *section, const char *key, int value)

  Writes an integer value to a section and key in an open configuration file.

- int **ggz_conf_write_list** (int handle, const char *section, const char *key, int argc, char **argv)

  Writes a list of string values to a section and key in an open configuration file.

- char * **ggz_conf_read_string** (int handle, const char *section, const char *key, const char *def)

  Reads a string value from an open configuration file.

- int **ggz_conf_read_int** (int handle, const char *section, const char *key, int def)

  Reads an integer value from an open configuration file.

- int **ggz_conf_read_list** (int handle, const char *section, const char *key, int *argcp, char ***argvp)

  Reads a list of string values from an open configuration file.

- int **ggz_conf_remove_section** (int handle, const char *section)

  This will remove an entire section and all its associated keys from a configuration file.

- int **ggz_conf_remove_key** (int handle, const char *section, const char *key)

  This will remove a single key from a configuration file.

- int **ggz_conf_get_sections** (int handle, int *argcp, char ***argvp)

  This function returns a list of all sections in a config file.

- int **ggz_conf_get_keys** (int handle, const char *section, int *argcp, char ***argvp)

  This function returns a list of all keys within a section in a config file.

- **GGZList** * **ggz_list_create** (**ggzEntryCompare** compare_func, **ggzEntryCreate** create_func, **ggzEntryDestroy** destroy_func, int options)

  *Create a new **GGZList** (p. 24).*

- int **ggz_list_insert** (**GGZList** *list, void *data)

  Insert data into a list.

- **GGZListEntry** * **ggz_list_head** (**GGZList** *list)

  Get the first node of a list.

- **GGZListEntry** * **ggz_list_tail** (**GGZList** *list)

  Get the last node of a list.

- **GGZListEntry** * **ggz_list_next** (**GGZListEntry** *entry)

  Get the next node of a list.

- **GGZListEntry** * **ggz_list_prev** (**GGZListEntry** *entry)

  Get the previous node of a list.

- void * **ggz_list_get_data** (**GGZListEntry** *entry)

  Retrieve the data stored in a list entry.

- **GGZListEntry** * **ggz_list_search** (**GGZList** *list, void *data)

  Search for a specified data item in the list.

- **GGZListEntry** * **ggz_list_search_alt** (**GGZList** *list, void *data, **ggzEntryCompare** compare_func)

Search for a specified data item in the list using a provided comparison function.

- void **ggz_list_delete_entry** (**GGZList** *list, **GGZListEntry** *entry)

  Removes an entry from a list, calling a destructor if registered.

- void **ggz_list_free** (**GGZList** *list)

  Free all resources associated with a list.

- int **ggz_list_count** (**GGZList** *list)

  Get the length of the list.

- int **ggz_list_compare_str** (void *a, void *b)

  Compare two character strings.

- void * **ggz_list_create_str** (void *data)

  Copy a character string.

- void **ggz_list_destroy_str** (void *data)

  Free a character string.

- **GGZStack** * **ggz_stack_new** (void)

  Create a new stack.

- void **ggz_stack_push** (**GGZStack** *stack, void *data)

  Push a data item onto the top of the stack.

- void * **ggz_stack_pop** (**GGZStack** *stack)

  Pop the top item off of the stack.

- void * **ggz_stack_top** (**GGZStack** *stack)

  Get the top item on the stack without popping it.

- void **ggz_stack_free** (**GGZStack** *stack)

  Free the stack.

- **GGZXMLElement** * **ggz_xmlelement_new** (const char *tag, const char *const *attrs, void(*process)(void *, **GGZXMLElement** *), void(*free)(**GGZXMLElement** *))

  *Create a new **GGZXMLElement** (p. 75) element.*

- void **ggz_xmlelement_init** (**GGZXMLElement** *element, const char *tag, const char *const *attrs, void(*process)(void *, **GGZXMLElement** *), void(*free)(**GGZXMLElement** *))

  *Initialize a **GGZXMLElement** (p. 75).*

- void **ggz_xmlelement_set_data** (**GGZXMLElement** *element, void *data)

  *Set ancillary data on a **GGZXMLElement** (p. 75) object.*

- const char * **ggz_xmlelement_get_tag** (**GGZXMLElement** *element)

  Get an XML element's name.

- const char * **ggz_xmlelement_get_attr** (**GGZXMLElement** *element, const char *attr)

  Get the value of an attribute on XML element.

- void * **ggz_xmlelement_get_data** (**GGZXMLElement** *element)

  Get the user-supplied data associated with an XML element.

- char * **ggz_xmlelement_get_text** (**GGZXMLElement** *element)

  Get an XML element's content text.

- void **ggz_xmlelement_add_text** (**GGZXMLElement** *element, const char *text, int len)

  Append a string to the element's content text.

- void **ggz_xmlelement_free** (**GGZXMLElement** *element)

  Free the memory associated with an XML element.

- void **ggz_debug_init** (const char **types, const char *file)

  Initialize and configure debugging for the program.

- **GGZDebugHandlerFunc ggz_debug_set_func** (**GGZDebugHandlerFunc** func)

  Set the debug handler function.

- void **ggz_debug_enable** (const char *type)

  Enable a specific type of debugging.

- void **ggz_debug_disable** (const char *type)

  Disable a specific type of debugging.

- void **ggz_debug** (const char *type, const char *fmt,...) ggz__attribute((format(printf

  Log a debugging message.

- void void **ggz_log** (const char *type, const char *fmt,...) ggz__attribute((format(printf

  Log a notice message.

- void void void **ggz_error_sys** (const char *fmt,...) ggz__attribute((format(printf

  Log a syscall error.

- void void void void **ggz_error_sys_exit** (const char *fmt,...) ggz__attribute((format(printf

  Log a fatal syscall error.

- void void void void **ggz__attribute** ((noreturn))
- void **ggz_error_msg** (const char *fmt,...) ggz__attribute((format(printf

  Log an error message.

- void void **ggz_error_msg_exit** (const char *fmt,...) ggz__attribute((format(printf

  Log a fatal error message.

- void **ggz_debug_cleanup** (**GGZCheckType** check)

Cleans up debugging state and prepares for exit.

- char * **ggz_xml_escape** (const char *str)

  Escape XML characters in a text string.

- char * **ggz_xml_unescape** (const char *str)

  Restore escaped XML characters into a text string.

- **GGZFile** * **ggz_get_file_struct** (int fdes)

  Setup a file structure to use with ***ggz_read_line()*** *(p. 43).*

- int **ggz_make_path** (const char *full)

  Create directories to fill out a path.

- char * **ggz_read_line** (**GGZFile** *file)

  Read a line of arbitrary length from a file.

- void **ggz_free_file_struct** (**GGZFile** *file)

  Deallocate a file structure allocated via ***ggz_get_file_struct()*** *(p. 42).*

- int **ggz_strcmp** (const char *s1, const char *s2)

  String comparison function that is safe with NULLs.

- int **ggz_strcasecmp** (const char *s1, const char *s2)

  Case-insensitive string comparison function that is safe with NULLs The function returns an integer less than, equal to, or greater than zero if s1 is found, respectively, to be less than, to match, or be greater than s2.

- int **ggz_set_io_error_func** (**ggzIOError** func)

  Set the ggz/easysock error handling function.

- **ggzIOError** **ggz_remove_io_error_func** (void)

  Remove the ggz/easysock error handling function.

- int **ggz_set_io_exit_func** (**ggzIOExit** func)

  Set the ggz/easysock exit function.

- **ggzIOExit** **ggz_remove_io_exit_func** (void)

  Remove the ggz/easysock exit function.

- unsigned int **ggz_get_io_alloc_limit** (void)

  Get libggz's limit on memory allocation.

- unsigned int **ggz_set_io_alloc_limit** (const unsigned int limit)

  Set libggz's limit on memory allocation.

- int **ggz_init_network** (void)

  Initialize the network.

- int **ggz_set_network_notify_func** (**ggzNetworkNotify** func)

Set the ggz/easysock resolver notification function.

- const char * **ggz_resolvename** (const char *name)

  Resolve a host name.

- const char * **ggz_getpeername** (int fd, int resolve)

  Get the IP address or host name of a connected peer.

- int **ggz_make_socket** (const **GGZSockType** type, const unsigned short port, const char *server)

  Make a socket connection.

- int **ggz_make_socket_or_die** (const **GGZSockType** type, const unsigned short port, const char *server)

  Make a socket connection, exiting on error.

- int **ggz_make_unix_socket** (const **GGZSockType** type, const char *name)

  Connect to a unix domain socket.

- int **ggz_make_unix_socket_or_die** (const **GGZSockType** type, const char *name)

  Connect to a unix domain socket, exiting on error.

- int **ggz_write_char** (const int sock, const char data)

  Write a character value to the given socket.

- void **ggz_write_char_or_die** (const int sock, const char data)

  Write a character value to the given socket, exiting on error.

- int **ggz_read_char** (const int sock, char *data)

  Read a character value from the given socket.

- void **ggz_read_char_or_die** (const int sock, char *data)

  Read a character value from the given socket, exiting on error.

- int **ggz_write_int** (const int sock, const int data)

  Write an integer to the socket in network byte order.

- void **ggz_write_int_or_die** (const int sock, const int data)

  Write an integer to the socket, exiting on error.

- int **ggz_read_int** (const int sock, int *data)

  Read an integer from the socket in network byte order.

- void **ggz_read_int_or_die** (const int sock, int *data)

  Read an integer from the socket, exiting on error.

- int **ggz_write_string** (const int sock, const char *data)

  Write a string to the given socket.

- void **ggz_write_string_or_die** (const int sock, const char *data)

Write a string to the given socket, exiting on error.

- int **ggz_va_write_string** (const int sock, const char *fmt,...) ggz__attribute((format(printf
  Write a printf-style formatted string to the given socket.
- int void **ggz_va_write_string_or_die** (const int sock, const char *fmt,...) ggz__attribute((format(printf
  Write a formatted string to the socket, exiting on error.
- int void int **ggz_read_string** (const int sock, char *data, const unsigned int len)
  Read a string from the given socket.
- void **ggz_read_string_or_die** (const int sock, char *data, const unsigned int len)
  Read a string from the given socket, exiting on error.
- int **ggz_read_string_alloc** (const int sock, char **data)
  Read and allocate a string from the given socket.
- void **ggz_read_string_alloc_or_die** (const int sock, char **data)
  Read and allocate string from the given socket, exiting on error.
- int **ggz_write_fd** (const int sock, int sendfd)
  Write a file descriptor to the given (local) socket.
- int **ggz_read_fd** (const int sock, int *recvfd)
  Read a file descriptor from the given (local) socket.
- int **ggz_writen** (const int sock, const void *vdata, size_t n)
  Write a sequence of bytes to the socket.
- int **ggz_readn** (const int sock, void *data, size_t n)
  Read a sequence of bytes from the socket.
- **hash_t ggz_hash_create** (const char *algo, const char *text)
  Create a hash over a text.
- **hash_t ggz_hmac_create** (const char *algo, const char *text, const char *secret)
  Create a HMAC hash over a text.
- char * **ggz_base16_encode** (const char *text, int length)
  Encodes text to base16.
- char * **ggz_base64_encode** (const char *text, int length)
  Encodes text to base64.
- char * **ggz_base64_decode** (const char *text, int length)
  Decodes text from base64.
- void **ggz_tls_init** (const char *certfile, const char *keyfile, const char *password)

Initialize TLS support on the server side.

- int **ggz_tls_support_query** (void)

  Check TLS support.

- const char * **ggz_tls_support_name** (void)

  Name of the TLS implementation.

- int **ggz_tls_enable_fd** (int fdes, **GGZTLSType** whoami, **GGZTLSVerificationType** verify)

  Enable TLS for a file descriptor.

- int **ggz_tls_disable_fd** (int fdes)

  Disable TLS for a file descriptor.

- size_t **ggz_tls_write** (int fd, void *ptr, size_t n)

  Write some bytes to a secured file descriptor.

- size_t **ggz_tls_read** (int fd, void *ptr, size_t n)

  Read from a secured file descriptor.

7.1.1 Detailed Description

Author:

Brent M.

Hendricks

Date:

Fri Nov 2 23:32:17 2001

Id

ggz.h (p. 73) 9362 2007-11-17 15:33:52Z josef

Header file for ggz components lib

7.1.2 Define Documentation

7.1.2.1 #define ggz__attribute(att)

Allow easy use of GCC's "attribute" macro for debugging.

Under gcc, we use the __attribute__ macro to check variadic arguments, for instance to printf-style functions. Other compilers may be able to do something similar, but this is generally unnecessary since it's only realy purpose is to give warning messages when the developer compiles the code.

Chapter 8

LibGGZ Page Documentation

8.1 Bug List

Group conf (p. **14**) The maximum length of a configuration file line is fixed at 1024 characters. Exceeding this length will result in parsing errors when the file is parsed. No internal problems should (hopefully) result, but values will be lost.

8.2 Todo List

Global ggz_set_io_error_func (p. **49**) Shouldn't this function return a void or ggz-IOError?

Global ggz_set_io_exit_func (p. **50**) Shouldn't this return a void?

Global ggz_set_network_notify_func (p. **51**) Shouldn't this return a void? (from ggz_-set_io_exit_func)

Global ggz_resolvename (p. **52**) Should this resolve synchronously in the special cases above?

Index

_GGZList, 67
    entries, 68
_GGZListEntry, 69
_GGZXMLElement, 70
_ggz_free
    memory, 12
_ggz_malloc
    memory, 11
_ggz_realloc
    memory, 12
_ggz_strdup
    memory, 12

conf
    ggz_conf_cleanup, 16
    ggz_conf_close, 16
    ggz_conf_commit, 17
    GGZ_CONF_CREATE, 16
    GGZ_CONF_DEBUG, 16
    ggz_conf_get_keys, 20
    ggz_conf_get_sections, 20
    ggz_conf_parse, 16
    GGZ_CONF_RDONLY, 16
    GGZ_CONF_RDWR, 16
    ggz_conf_read_int, 19
    ggz_conf_read_list, 19
    ggz_conf_read_string, 18
    ggz_conf_remove_key, 20
    ggz_conf_remove_section, 19
    ggz_conf_write_int, 17
    ggz_conf_write_list, 18
    ggz_conf_write_string, 17
    GGZConfType, 16
Configuration file parsing, 14

debug
    GGZ_CHECK_MEM, 37
    GGZ_CHECK_NONE, 37
    ggz_debug, 39
    ggz_debug_cleanup, 40
    ggz_debug_disable, 38
    ggz_debug_enable, 38
    ggz_debug_init, 37
    ggz_debug_set_func, 38
    ggz_error_msg, 40
    ggz_error_msg_exit, 40
    ggz_error_sys, 39
    ggz_error_sys_exit, 39
    ggz_log, 39
    GGZCheckType, 37
    GGZDebugHandlerFunc, 37
Debug/error logging, 36

easysock
    GGZ_DATA_CHAR, 49
    GGZ_DATA_FD, 49
    GGZ_DATA_INT, 49
    GGZ_DATA_NONE, 49
    GGZ_DATA_STRING, 49
    ggz_get_io_alloc_limit, 50
    ggz_getpeername, 52
    ggz_init_network, 51
    GGZ_IO_ALLOCATE, 49
    GGZ_IO_CREATE, 49
    GGZ_IO_READ, 49
    GGZ_IO_WRITE, 49
    ggz_make_socket, 52
    ggz_make_socket_or_die, 53
    ggz_make_unix_socket, 53
    ggz_make_unix_socket_or_die, 53
    ggz_read_char, 54
    ggz_read_char_or_die, 54
    ggz_read_fd, 58
    ggz_read_int, 55
    ggz_read_int_or_die, 55
    ggz_read_string, 56
    ggz_read_string_alloc, 57
    ggz_read_string_alloc_or_die, 57
    ggz_read_string_or_die, 57
    ggz_readn, 59
    ggz_remove_io_error_func, 50
    ggz_remove_io_exit_func, 50
    ggz_resolvename, 52
    ggz_set_io_alloc_limit, 51
    ggz_set_io_error_func, 49
    ggz_set_io_exit_func, 50
    ggz_set_network_notify_func, 51
    GGZ_SOCK_CLIENT, 49
    GGZ_SOCK_SERVER, 49
    GGZ_SOCKET_DEBUG, 48

ggz_va_write_string, 56
ggz_va_write_string_or_die, 56
ggz_write_char, 54
ggz_write_char_or_die, 54
ggz_write_fd, 58
ggz_write_int, 55
ggz_write_int_or_die, 55
ggz_write_string, 56
ggz_write_string_or_die, 56
ggz_writen, 58
GGZDataType, 49
ggzIOError, 48
ggzIOExit, 48
GGZIOType, 49
ggzNetworkNotify, 48
GGZSockType, 49
Easysock IO, 45
entries
_GGZList, 68

ggz.h, 73
ggz__attribute, 84
ggz__attribute
ggz.h, 84
ggz_base16_encode
security, 62
ggz_base64_decode
security, 63
ggz_base64_encode
security, 62
GGZ_CHECK_MEM
debug, 37
GGZ_CHECK_NONE
debug, 37
ggz_conf_cleanup
conf, 16
ggz_conf_close
conf, 16
ggz_conf_commit
conf, 17
GGZ_CONF_CREATE
conf, 16
GGZ_CONF_DEBUG
conf, 16
ggz_conf_get_keys
conf, 20
ggz_conf_get_sections
conf, 20
ggz_conf_parse
conf, 16
GGZ_CONF_RDONLY
conf, 16
GGZ_CONF_RDWR
conf, 16
ggz_conf_read_int
conf, 19
ggz_conf_read_list
conf, 19
ggz_conf_read_string
conf, 18
ggz_conf_remove_key
conf, 20
ggz_conf_remove_section
conf, 19
ggz_conf_write_int
conf, 17
ggz_conf_write_list
conf, 18
ggz_conf_write_string
conf, 17
GGZ_DATA_CHAR
easysock, 49
GGZ_DATA_FD
easysock, 49
GGZ_DATA_INT
easysock, 49
GGZ_DATA_NONE
easysock, 49
GGZ_DATA_STRING
easysock, 49
ggz_debug
debug, 39
ggz_debug_cleanup
debug, 40
ggz_debug_disable
debug, 38
ggz_debug_enable
debug, 38
ggz_debug_init
debug, 37
ggz_debug_set_func
debug, 38
ggz_error_msg
debug, 40
ggz_error_msg_exit
debug, 40
ggz_error_sys
debug, 39
ggz_error_sys_exit
debug, 39
ggz_free
memory, 11
ggz_free_file_struct
misc, 43
ggz_get_file_struct
misc, 42
ggz_get_io_alloc_limit
easysock, 50

ggz_getpeername
easysock, 52
ggz_hash_create
security, 62
ggz_hmac_create
security, 62
ggz_init_network
easysock, 51
GGZ_IO_ALLOCATE
easysock, 49
GGZ_IO_CREATE
easysock, 49
GGZ_IO_READ
easysock, 49
GGZ_IO_WRITE
easysock, 49
ggz_list_compare_str
list, 28
ggz_list_count
list, 28
ggz_list_create
list, 25
ggz_list_create_str
list, 28
ggz_list_delete_entry
list, 27
ggz_list_destroy_str
list, 29
ggz_list_free
list, 28
ggz_list_get_data
list, 26
ggz_list_head
list, 26
ggz_list_insert
list, 25
ggz_list_next
list, 26
ggz_list_prev
list, 26
ggz_list_search
list, 27
ggz_list_search_alt
list, 27
ggz_list_tail
list, 26
ggz_log
debug, 39
ggz_make_path
misc, 42
ggz_make_socket
easysock, 52
ggz_make_socket_or_die
easysock, 53
ggz_make_unix_socket
easysock, 53
ggz_make_unix_socket_or_die
easysock, 53
ggz_malloc
memory, 10
GGZ_MEM_DEBUG
memory, 10
ggz_memory_check
memory, 13
ggz_read_char
easysock, 54
ggz_read_char_or_die
easysock, 54
ggz_read_fd
easysock, 58
ggz_read_int
easysock, 55
ggz_read_int_or_die
easysock, 55
ggz_read_line
misc, 43
ggz_read_string
easysock, 56
ggz_read_string_alloc
easysock, 57
ggz_read_string_alloc_or_die
easysock, 57
ggz_read_string_or_die
easysock, 57
ggz_readn
easysock, 59
ggz_realloc
memory, 10
ggz_remove_io_error_func
easysock, 50
ggz_remove_io_exit_func
easysock, 50
ggz_resolvename
easysock, 52
ggz_set_io_alloc_limit
easysock, 51
ggz_set_io_error_func
easysock, 49
ggz_set_io_exit_func
easysock, 50
ggz_set_network_notify_func
easysock, 51
GGZ_SOCK_CLIENT
easysock, 49
GGZ_SOCK_SERVER
easysock, 49
GGZ_SOCKET_DEBUG
easysock, 48

ggz_stack_free
stack, 31
ggz_stack_new
stack, 30
ggz_stack_pop
stack, 30
ggz_stack_push
stack, 30
ggz_stack_top
stack, 31
ggz_strcasecmp
misc, 43
ggz_strcmp
misc, 43
ggz_strdup
memory, 11
ggz_strncpy
memory, 13
GGZ_TLS_CLIENT
security, 61
ggz_tls_disable_fd
security, 64
ggz_tls_enable_fd
security, 64
ggz_tls_init
security, 63
ggz_tls_read
security, 65
GGZ_TLS_SERVER
security, 61
ggz_tls_support_name
security, 64
ggz_tls_support_query
security, 63
GGZ_TLS_VERIFY_NONE
security, 61
GGZ_TLS_VERIFY_PEER
security, 61
ggz_tls_write
security, 64
ggz_va_write_string
easysock, 56
ggz_va_write_string_or_die
easysock, 56
ggz_write_char
easysock, 54
ggz_write_char_or_die
easysock, 54
ggz_write_fd
easysock, 58
ggz_write_int
easysock, 55
ggz_write_int_or_die
easysock, 55
ggz_write_string
easysock, 56
ggz_write_string_or_die
easysock, 56
ggz_writen
easysock, 58
ggz_xml_escape
misc, 41
ggz_xml_unescape
misc, 42
ggz_xmlelement_add_text
xml, 35
ggz_xmlelement_free
xml, 35
ggz_xmlelement_get_attr
xml, 34
ggz_xmlelement_get_data
xml, 34
ggz_xmlelement_get_tag
xml, 34
ggz_xmlelement_get_text
xml, 34
ggz_xmlelement_init
xml, 33
ggz_xmlelement_new
xml, 33
ggz_xmlelement_set_data
xml, 33
GGZCheckType
debug, 37
GGZConfType
conf, 16
GGZDataType
easysock, 49
GGZDebugHandlerFunc
debug, 37
ggzEntryCompare
list, 24
ggzEntryCreate
list, 24
ggzEntryDestroy
list, 24
ggzIOError
easysock, 48
ggzIOExit
easysock, 48
GGZIOType
easysock, 49
GGZList
list, 24
GGZListEntry
list, 24
ggzNetworkNotify
easysock, 48

GGZSockType
  easysock, 49
GGZTLSType
  security, 61
GGZTLSVerificationType
  security, 61

hash_t, 71

list
  ggz_list_compare_str, 28
  ggz_list_count, 28
  ggz_list_create, 25
  ggz_list_create_str, 28
  ggz_list_delete_entry, 27
  ggz_list_destroy_str, 29
  ggz_list_free, 28
  ggz_list_get_data, 26
  ggz_list_head, 26
  ggz_list_insert, 25
  ggz_list_next, 26
  ggz_list_prev, 26
  ggz_list_search, 27
  ggz_list_search_alt, 27
  ggz_list_tail, 26
  ggzEntryCompare, 24
  ggzEntryCreate, 24
  ggzEntryDestroy, 24
  GGZList, 24
  GGZListEntry, 24
List functions, 22

memory
  _ggz_free, 12
  _ggz_malloc, 11
  _ggz_realloc, 12
  _ggz_strdup, 12
  ggz_free, 11
  ggz_malloc, 10
  GGZ_MEM_DEBUG, 10
  ggz_memory_check, 13
  ggz_realloc, 10
  ggz_strdup, 11
  ggz_strncpy, 13
Memory Handling, 9
misc
  ggz_free_file_struct, 43
  ggz_get_file_struct, 42
  ggz_make_path, 42
  ggz_read_line, 43
  ggz_strcasecmp, 43
  ggz_strcmp, 43
  ggz_xml_escape, 41
  ggz_xml_unescape, 42
Miscellaneous convenience functions, 41

security
  ggz_base16_encode, 62
  ggz_base64_decode, 63
  ggz_base64_encode, 62
  ggz_hash_create, 62
  ggz_hmac_create, 62
  GGZ_TLS_CLIENT, 61
  ggz_tls_disable_fd, 64
  ggz_tls_enable_fd, 64
  ggz_tls_init, 63
  ggz_tls_read, 65
  GGZ_TLS_SERVER, 61
  ggz_tls_support_name, 64
  ggz_tls_support_query, 63
  GGZ_TLS_VERIFY_NONE, 61
  GGZ_TLS_VERIFY_PEER, 61
  ggz_tls_write, 64
  GGZTLSType, 61
  GGZTLSVerificationType, 61
Security functions, 60
stack
  ggz_stack_free, 31
  ggz_stack_new, 30
  ggz_stack_pop, 30
  ggz_stack_push, 30
  ggz_stack_top, 31
Stacks, 30

xml
  ggz_xmlelement_add_text, 35
  ggz_xmlelement_free, 35
  ggz_xmlelement_get_attr, 34
  ggz_xmlelement_get_data, 34
  ggz_xmlelement_get_tag, 34
  ggz_xmlelement_get_text, 34
  ggz_xmlelement_init, 33
  ggz_xmlelement_new, 33
  ggz_xmlelement_set_data, 33
XML parsing, 32