# Btparser

A program failure analysis library

Karel Klíč

August 27, 2012

# Contents

# Chapter 1

# Overview

# Part I

# Concepts

# Part II

# Implementation

# Chapter 2

# Overview

Btparser is implemented in the C language as defined in the C99 standard (ISO/IEC 9899:1999). It uses the C standard library as well as some additional libraries:

- elfutils

# Chapter 3

# Data Structure Index

## 3.1 Data Structures

Here are the data structures with brief descriptions:



## 3.2 File List

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

# Chapter 4

# Data Structure Documentation

## 4.1 btp_callgraph Struct Reference

Collaboration diagram for btp_callgraph:


### Data Fields

- uint64_t **address**
- uint64_t * **callees**
- struct btp_callgraph * **next**

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

- callgraph.h

## 4.2 btp_cluster Struct Reference

`#include <cluster.h>`Collaboration diagram for btp_cluster:


### Data Fields

- int **size**
- int * **objects**
- struct btp_cluster * **next**

### 4.2.1 Detailed Description

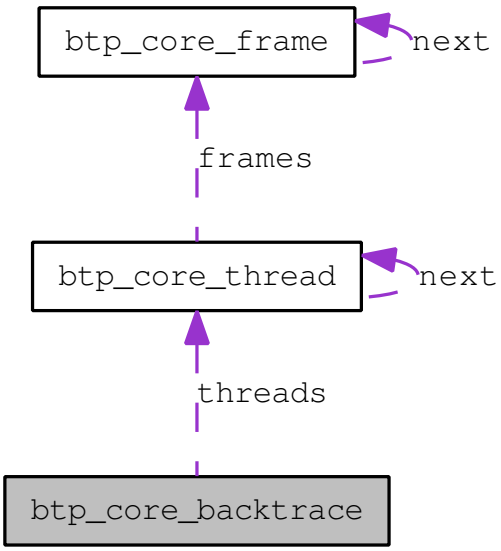
Represents a cluster of objects.

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

- cluster.h

## 4.3 btp_core_backtrace Struct Reference

Collaboration diagram for btp_core_backtrace:




### Data Fields

- enum btp_core_backtrace_type **type**
- struct btp_core_thread * **threads**

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

- core_backtrace.h

## 4.4 btp_core_frame Struct Reference

`#include <core_frame.h>`Collaboration diagram for btp_core_frame:


### Data Fields

- uint64_t address
- char * build_id
- uint64_t **build_id_offset**
- char * **function_name**
- char * **file_name**
- char * fingerprint
- struct btp_core_frame * next

### 4.4.1 Detailed Description

A frame representing a function call on a call stack of a thread.

### 4.4.2 Field Documentation

#### 4.4.2.1 uint64_t btp_core_frame::address

Address of the machine code in memory. This is useful only when build_id is not present for some reason. For example, this might be a null dereference (address is 0) or calling a method from null class pointer (address is a low number -- offset to the class).

Some programs generate machine code during runtime (JavaScript engines, JVM, the Gallium llvmpipe driver).

#### 4.4.2.2 char* btp_core_frame::build_id

Build id of the ELF binary. It might be NULL if the frame does not point to memory with code.

#### 4.4.2.3 char* btp_core_frame::fingerprint

Hash of the function contents.

#### 4.4.2.4 struct btp_core_frame* btp_core_frame::next [read]

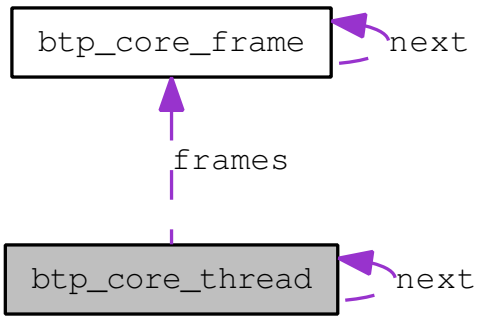
A sibling frame residing below this one, or NULL if this is the last frame in the parent thread.

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

- core_frame.h

## 4.5 btp_core_thread Struct Reference

`#include <core_thread.h>`Collaboration diagram for btp_core_thread:




### Data Fields

- struct btp_core_frame ∗ frames
- struct btp_core_thread ∗ next

### 4.5.1 Detailed Description

Represents a thread containing frames.

### 4.5.2 Field Documentation

#### 4.5.2.1 struct btp_core_frame∗ btp_core_thread::frames [read]

Thread's frames, starting from the top of the stack.

#### 4.5.2.2 struct btp_core_thread∗ btp_core_thread::next [read]

A sibling thread, or NULL if this is the last thread in a backtrace.

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

- core_thread.h

## 4.6 btp_dendrogram Struct Reference

`#include <cluster.h>`

### Data Fields

- int **size**
- int * **order**
- float * merge_levels

### 4.6.1 Detailed Description

Represents a dendrogram created by clustering.

### 4.6.2 Field Documentation

#### 4.6.2.1 float* btp_dendrogram::merge_levels

Levels at which the clusters were merged. The clustering can be reconstructed in order of increasing levels. There are (size - 1) levels.

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

- cluster.h

## 4.7 btp_disasm_state Struct Reference

### Data Fields

- bfd * **bfd_file**
- disassembler_ftype **disassembler**
- struct disassemble_info **info**
- char * **error_message**

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

- disassembler.h

## 4.8 btp_distances Struct Reference

`#include <metrics.h>`

### Data Fields

- int **m**
- int **n**
- float * **distances**

### 4.8.1 Detailed Description

Represents an m-by-n distance matrix. (only entries (i, j) where i < j are actually stored)

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

- metrics.h

## 4.9 btp_elf_frame_description_entry Struct Reference

`#include <elves.h>`Collaboration diagram for btp_elf_frame_description_entry:


### Data Fields

- uint64_t start_address
- uint64_t length
- struct btp_elf_frame_description_entry * **next**

### 4.9.1 Detailed Description

A Frame Description Entry (FDE) representing items in the .eh_frame section in ELF binaries.

### 4.9.2 Field Documentation

#### 4.9.2.1 uint64_t btp_elf_frame_description_entry::length

Length of the function in bytes.

#### 4.9.2.2 uint64_t btp_elf_frame_description_entry::start_address

Offset where a function starts. If the function is present in the Procedure Linkage Table, this address matches some address in btp_elf_plt_entry.

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

- elves.h

## 4.10 btp_elf_plt_entry Struct Reference

`#include <elves.h>`Collaboration diagram for btp_elf_plt_entry:


### Data Fields

- uint64_t address
- char * symbol_name
- struct btp_elf_plt_entry * **next**

### 4.10.1 Detailed Description

File name elf.h cannot be used due to collision with <elf.h> system include. An entry of the Procedure Linkage Table (PLT).

### 4.10.2 Field Documentation

#### 4.10.2.1 uint64_t btp_elf_plt_entry::address

Address of the entry.

#### 4.10.2.2 char* btp_elf_plt_entry::symbol_name

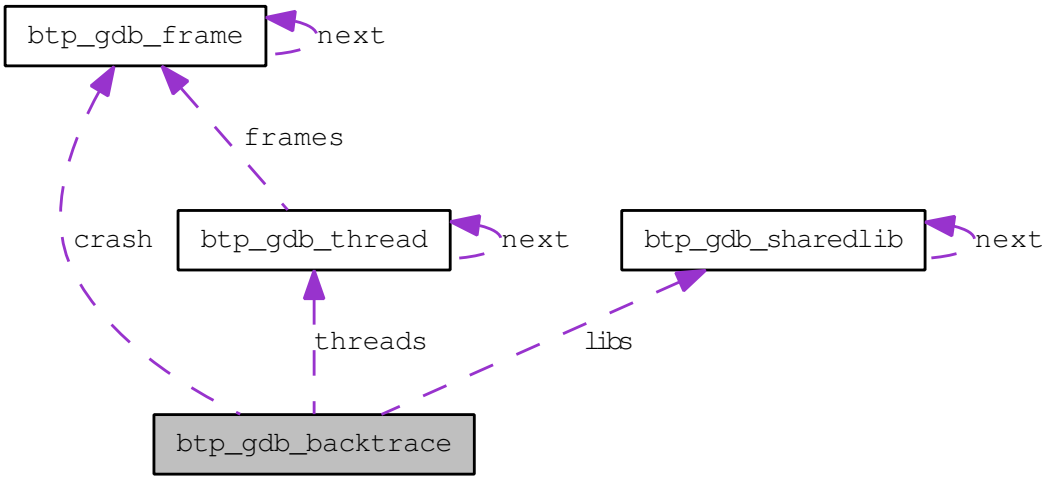
Symbol name corresponding to the address.

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

- elves.h

## 4.11 btp_gdb_backtrace Struct Reference

`#include <gdb_backtrace.h>`Collaboration diagram for btp_gdb_backtrace:




### Data Fields

- struct btp_gdb_thread ∗ **threads**
- struct btp_gdb_frame ∗ crash
- struct btp_gdb_sharedlib ∗ libs

### 4.11.1 Detailed Description

A backtrace obtained at the time of a program crash, consisting of several threads which contains frames.

This structure represents a backtrace as produced by the GNU Debugger.

### 4.11.2 Field Documentation

#### 4.11.2.1 struct btp_gdb_frame∗ btp_gdb_backtrace::crash [read]

The frame where the crash happened according to debugger. It might be that we can not tell to which thread this frame belongs, because some threads end with mutually indistinguishable frames.

#### 4.11.2.2 struct btp_gdb_sharedlib∗ btp_gdb_backtrace::libs [read]

Shared libraries loaded at the moment of crash.

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

- gdb_backtrace.h

## 4.12 btp_gdb_frame Struct Reference

`#include <gdb_frame.h>`Collaboration diagram for btp_gdb_frame:


### Data Fields

- char * function_name
- char * function_type
- unsigned number
- char * source_file
- unsigned source_line
- bool signal_handler_called
- uint64_t address
- char * library_name
- struct btp_gdb_frame * next

### 4.12.1 Detailed Description

A frame representing a function call or a signal handler on a call stack of a thread.

### 4.12.2 Field Documentation

#### 4.12.2.1 uint64_t btp_gdb_frame::address

The function address in the computer memory, or -1 when the address is unknown. Address is unknown when the frame represents inlined function.

#### 4.12.2.2 char* btp_gdb_frame::function_name

A function name or NULL. If it's NULL, signal_handler_called is true.

#### 4.12.2.3 char* btp_gdb_frame::function_type

A function type, or NULL if it isn't present.

#### 4.12.2.4 char* btp_gdb_frame::library_name

A library name or NULL.

#### 4.12.2.5 struct btp_gdb_frame* btp_gdb_frame::next [read]

A sibling frame residing below this one, or NULL if this is the last frame in the parent thread.

#### 4.12.2.6 unsigned btp_gdb_frame::number

A frame number in a thread. It does not necessarily show the actual position in the thread, as this number is set by the parser and never updated.

#### 4.12.2.7 bool btp_gdb_frame::signal_handler_called

Signal handler was called on this frame.

#### 4.12.2.8 char* btp_gdb_frame::source_file

The name of the source file containing the function definition, or the name of the binary file (.so) with the binary code of the function, or NULL.

#### 4.12.2.9 unsigned btp_gdb_frame::source_line

A line number in the source file, determining the position of the function definition, or -1 when unknown.

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

- gdb_frame.h

## 4.13 btp_gdb_sharedlib Struct Reference

Collaboration diagram for btp_gdb_sharedlib:


### Data Fields

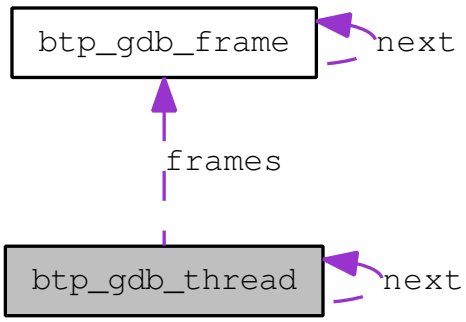
- uint64_t **from**
- uint64_t **to**
- int **symbols**
- char * **soname**
- struct btp_gdb_sharedlib * **next**

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

- gdb_sharedlib.h

## 4.14 btp_gdb_thread Struct Reference

`#include <gdb_thread.h>`Collaboration diagram for btp_gdb_thread:




### Data Fields

- unsigned **number**
- struct btp_gdb_frame ∗ frames
- struct btp_gdb_thread ∗ next

### 4.14.1 Detailed Description

Represents a thread containing frames.

### 4.14.2 Field Documentation

#### 4.14.2.1 struct btp_gdb_frame∗ btp_gdb_thread::frames [read]

Thread's frames, starting from the top of the stack.

#### 4.14.2.2 struct btp_gdb_thread∗ btp_gdb_thread::next [read]

A sibling thread, or NULL if this is the last thread in a backtrace.

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

- gdb_thread.h

## 4.15 btp_location Struct Reference

`#include <location.h>`

### Data Fields

- int line
- int column
- const char * message

### 4.15.1 Detailed Description

A location in the backtrace file with an attached message. It's used for error reporting: the line and the column points to the place where a parser error occurred, and the message explains what the parser expected and didn't find on that place.

### 4.15.2 Field Documentation

#### 4.15.2.1 int btp_location::column

Starts from 0.

#### 4.15.2.2 int btp_location::line

Starts from 1.

#### 4.15.2.3 const char* btp_location::message

Error message related to the line and column. Do not release the memory this pointer points to.

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

- location.h

## 4.16 btp_sha1_state Struct Reference

### Data Fields

- uint8_t **wbuffer** [64]
- uint64_t **total64**
- uint32_t **hash** [8]

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

- sha1.h

## 4.17 btp_strbuf Struct Reference

### Data Fields

- int alloc
- int len
- char * **buf**

### 4.17.1 Field Documentation

#### 4.17.1.1 int btp_strbuf::alloc

Size of the allocated buffer. Always > 0.

#### 4.17.1.2 int btp_strbuf::len

Length of the string, without the ending .

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

- strbuf.h

## 4.18 btp_unstrip_entry Struct Reference

`#include <unstrip.h>`Collaboration diagram for btp_unstrip_entry:


### Data Fields

- uint64_t **start**
- uint64_t **length**
- char * **build_id**
- char * **file_name**
- char * **mod_name**
- struct btp_unstrip_entry * **next**

### 4.18.1 Detailed Description

Output of the unstrip utility.

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

- unstrip.h

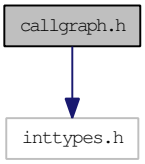
# Chapter 5

# File Documentation

## 5.1 callgraph.h File Reference

Call graph for ELF binaries. `#include <inttypes.h>`

Include dependency graph for callgraph.h:




### Data Structures

- struct btp_callgraph

### Functions

- struct btp_callgraph ∗ **btp_callgraph_compute** (struct btp_disasm_state ∗disassembler, struct btp_elf_frame_description_entry ∗fde_entries, char ∗∗error_message)
- struct btp_callgraph ∗ btp_callgraph_extend (struct btp_callgraph ∗callgraph, uint64_t start_address, struct btp_disasm_state ∗disassembler, struct btp_elf_frame_description_entry ∗fde_entries, char ∗∗error_message)
- void **btp_callgraph_free** (struct btp_callgraph ∗callgraph)
- struct btp_callgraph ∗ **btp_callgraph_find** (struct btp_callgraph ∗callgraph, uint64_t address)
- struct btp_callgraph ∗ **btp_callgraph_last** (struct btp_callgraph ∗callgraph)

### 5.1.1 Detailed Description

Call graph for ELF binaries. Call graph represents calling relationships between subroutines in ELF binaries. Only static relationships obtained from CALL-like instructions with numeric offsets are handled.

Call graph is used by fingerprinting algorithms.

### 5.1.2 Function Documentation

#### 5.1.2.1 struct btp_callgraph* btp_callgraph_extend (struct btp_callgraph * *callgraph*, uint64_t *start_address*, struct btp_disasm_state * *disassembler*, struct btp_elf_frame_description_entry * *fde_entries*, char ** *error_message*) [read]

Assumption: when a fde is included in the callgraph, we assume that all callees are included as well.

## 5.2 cluster.h File Reference

Clustering for stack trace threads.

### Data Structures

- struct btp_dendrogram
- struct btp_cluster

### Functions

- struct btp_dendrogram * btp_dendrogram_new (int size)
- void btp_dendrogram_free (struct btp_dendrogram *dendrogram)
- struct btp_dendrogram * btp_distances_cluster_objects (struct btp_distances *distances)
- struct btp_cluster * btp_cluster_new (int size)
- void btp_cluster_free (struct btp_cluster *cluster)
- struct btp_cluster * btp_dendrogram_cut (struct btp_dendrogram *dendrogram, float level, int min_size)

### 5.2.1 Detailed Description

Clustering for stack trace threads. The implemented clustering algorithm assigns a set of stack trace threads into groups. Each group represents a single program flaw.

### 5.2.2 Function Documentation

#### 5.2.2.1 void btp_cluster_free (struct btp_cluster * *cluster*)

Releases the memory held by the cluster.

**Parameters:**

***dendrogram*** If cluster is NULL, no operation is performed.

#### 5.2.2.2 struct btp_cluster* btp_cluster_new (int *size*) [read]

Creates and initializes a new cluster.

**Parameters:**

***size*** Number of objects in the cluster.

**Returns:**

It never returns NULL. The returned pointer must be released by btp_cluster_free().

#### 5.2.2.3 struct btp_cluster* btp_dendrogram_cut (struct btp_dendrogram * *dendrogram*, float *level*, int *min_size*) [read]

Cuts a dendrogram at specified level.

**Parameters:**

***dendrogram*** The dendrogram which should be cut. The structure is not modified by this call.

***level*** The cutting level of distance.

***min_size*** The minimum size of clusters which should be returned.

**Returns:**

List of clusters, NULL if empty.

#### 5.2.2.4 void btp_dendrogram_free (struct btp_dendrogram * *dendrogram*)

Releases the memory held by the dendrogram.

**Parameters:**

***dendrogram*** If dendrogram is NULL, no operation is performed.

#### 5.2.2.5 struct btp_dendrogram* btp_dendrogram_new (int *size*) [read]

Creates and initializes a new dendrogram structure.

**Parameters:**

***size*** Number of objects.

**Returns:**

It never returns NULL. The returned pointer must be released by btp_dendrogram_free().

#### 5.2.2.6 struct btp_dendrogram* btp_distances_cluster_objects (struct btp_distances * *distances*) [read]

Performs hierarchical agglomerative clustering on objects.

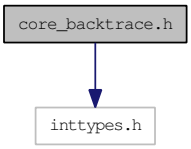
**Parameters:**

***distances*** Distances between the objects. The structure is not modified by calling this function.

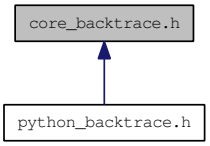
## 5.3 core_backtrace.h File Reference

Low-level stack trace generated directly from core dump. `#include <inttypes.h>`

Include dependency graph for core_backtrace.h:




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




### Data Structures

- struct btp_core_backtrace

### Enumerations

- enum **btp_core_backtrace_type** { **BTP_USERSPACE**, **BTP_PYTHON**, **BTP_KERNELOOPS** }

### Functions

- struct btp_core_backtrace * btp_core_backtrace_new ()
- void btp_core_backtrace_init (struct btp_core_backtrace *backtrace)
- void btp_core_backtrace_free (struct btp_core_backtrace *backtrace)
- struct btp_core_backtrace * btp_core_backtrace_dup (struct btp_core_backtrace *backtrace)
- int btp_core_backtrace_get_thread_count (struct btp_core_backtrace *backtrace)
- struct btp_core_backtrace * btp_core_backtrace_parse (const char **input, struct btp_location *location)
- char * btp_core_backtrace_to_text (struct btp_core_backtrace *backtrace)
- struct btp_core_backtrace * **btp_core_backtrace_create** (const char *gdb_backtrace_text, const char *unstrip_text, const char *executable_path)

### 5.3.1 Detailed Description

Low-level stack trace generated directly from core dump.

### 5.3.2 Function Documentation

#### 5.3.2.1 struct btp_core_backtrace* btp_core_backtrace_dup (struct btp_core_backtrace * *backtrace*) [read]

Creates a duplicate of the backtrace.

**Parameters:**

***backtrace*** The backtrace to be copied. It's not modified by this function.

**Returns:**

This function never returns NULL. The returned duplicate must be released by calling the function btp_core_backtrace_free().

#### 5.3.2.2 void btp_core_backtrace_free (struct btp_core_backtrace * *backtrace*)

Releases the memory held by the backtrace, its threads and frames.

**Parameters:**

***backtrace*** If the backtrace is NULL, no operation is performed.

#### 5.3.2.3 int btp_core_backtrace_get_thread_count (struct btp_core_backtrace * *backtrace*)

Returns a number of threads in the backtrace.

**Parameters:**

***backtrace*** It's not modified by calling this function.

#### 5.3.2.4 void btp_core_backtrace_init (struct btp_core_backtrace * *backtrace*)

Initializes all members of the backtrace structure to their default values. No memory is released, members are simply overwritten. This is useful for initializing a backtrace structure placed on the stack.

#### 5.3.2.5 struct btp_core_backtrace* btp_core_backtrace_new () [read]

Creates and initializes a new backtrace structure.

**Returns:**

It never returns NULL. The returned pointer must be released by calling the function btp_core_backtrace_free().

#### 5.3.2.6 struct btp_core_backtrace* btp_core_backtrace_parse (const char ** *input*, struct btp_location * *location*) [read]

Parses a textual backtrace and puts it into a structure. If parsing fails, the input parameter is not changed and NULL is returned.

**Note:**

Backtrace can be serialized to string via btp_core_backtrace_to_text().

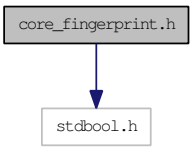
#### 5.3.2.7 char* btp_core_backtrace_to_text (struct btp_core_backtrace * *backtrace*)

Serializes backtrace to string. Newly allocated memory containing the textual representation of the provided backtrace. Caller should free the memory when it's no longer needed.

## 5.4 core_fingerprint.h File Reference

Fingerprint algorithm for core stack traces. `#include <stdbool.h>`

Include dependency graph for core_fingerprint.h:




### Functions

- bool **btp_core_fingerprint_generate** (struct btp_core_backtrace *backtrace, char **error_message)
- bool **btp_core_fingerprint_generate_for_binary** (struct btp_core_thread *thread, const char *binary_path, char **error_message)

### 5.4.1 Detailed Description

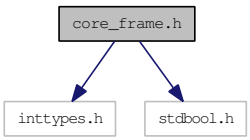
Fingerprint algorithm for core stack traces.

## 5.5 core_frame.h File Reference

Single frame of core stack trace thread. `#include <inttypes.h>`

`#include <stdbool.h>`

Include dependency graph for core_frame.h:




### Data Structures

- struct btp_core_frame

### Functions

- struct btp_core_frame * btp_core_frame_new ()
- void btp_core_frame_init (struct btp_core_frame *frame)
- void btp_core_frame_free (struct btp_core_frame *frame)
- struct btp_core_frame * btp_core_frame_dup (struct btp_core_frame *frame, bool siblings)
- int btp_core_frame_cmp (struct btp_core_frame *frame1, struct btp_core_frame *frame2)
- struct btp_core_frame * btp_core_frame_append (struct btp_core_frame *dest, struct btp_core_frame *item)
- void btp_core_frame_append_to_str (struct btp_core_frame *frame, struct btp_strbuf *dest)

### 5.5.1 Detailed Description

Single frame of core stack trace thread.

### 5.5.2 Function Documentation

#### 5.5.2.1 struct btp_core_frame* btp_core_frame_append (struct btp_core_frame * *dest*, struct btp_core_frame * *item*) [read]

Appends 'item' at the end of the list 'dest'.

**Returns:**

This function returns the 'dest' frame. If 'dest' is NULL, it returns the 'item' frame.

#### 5.5.2.2 void btp_core_frame_append_to_str (struct btp_core_frame * *frame*, struct btp_strbuf * *dest*)

Appends the textual representation of the frame to the string buffer.

**Parameters:**

***frame*** It must be a non-NULL pointer. It's not modified by calling this function.

#### 5.5.2.3 int btp_core_frame_cmp (struct btp_core_frame * *frame1*, struct btp_core_frame * *frame2*)

Compares two frames.

**Parameters:**

***frame1*** It must be non-NULL pointer. It's not modified by calling this function.

***frame2*** It must be non-NULL pointer. It's not modified by calling this function.

**Returns:**

Returns 0 if the frames are same. Returns negative number if frame1 is found to be 'less' than frame2. Returns positive number if frame1 is found to be 'greater' than frame2.

#### 5.5.2.4 struct btp_core_frame* btp_core_frame_dup (struct btp_core_frame * *frame*, bool *siblings*) [read]

Creates a duplicate of the frame.

**Parameters:**

***frame*** It must be non-NULL pointer. The frame is not modified by calling this function.

***siblings*** Whether to duplicate also siblings referenced by frame->next. If false, frame->next is not duplicated for the new frame, but it is set to NULL.

**Returns:**

This function never returns NULL. If the returned duplicate is not shallow, it must be released by calling the function btp_gdb_frame_free().

#### 5.5.2.5 void btp_core_frame_free (struct btp_core_frame * *frame*)

Releases the memory held by the frame. The frame siblings are not released.

**Parameters:**

***frame*** If the frame is NULL, no operation is performed.

#### 5.5.2.6 void btp_core_frame_init (struct btp_core_frame * *frame*)

Initializes all members of the frame structure to their default values. No memory is released, members are simply overwritten. This is useful for initializing a frame structure placed on the stack.

#### 5.5.2.7 struct btp_core_frame* btp_core_frame_new () [read]

Creates and initializes a new frame structure.

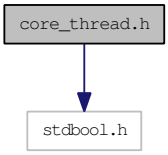
**Returns:**

It never returns NULL. The returned pointer must be released by calling the function btp_core_frame_-free().

## 5.6 core_thread.h File Reference

Single thread of execution of core stack trace. `#include <stdbool.h>`

Include dependency graph for core_thread.h:




### Data Structures

- struct btp_core_thread

### Functions

- struct btp_core_thread * btp_core_thread_new ()
- void btp_core_thread_init (struct btp_core_thread *thread)
- void btp_core_thread_free (struct btp_core_thread *thread)
- struct btp_core_thread * btp_core_thread_dup (struct btp_core_thread *thread, bool siblings)
- int btp_core_thread_cmp (struct btp_core_thread *thread1, struct btp_core_thread *thread2)
- struct btp_core_thread * btp_core_thread_append (struct btp_core_thread *dest, struct btp_core_thread *item)
- int btp_core_thread_get_frame_count (struct btp_core_thread *thread)
- void btp_core_thread_append_to_str (struct btp_core_thread *thread, struct btp_strbuf *dest)

### 5.6.1 Detailed Description

Single thread of execution of core stack trace.

### 5.6.2 Function Documentation

#### 5.6.2.1 struct btp_core_thread* btp_core_thread_append (struct btp_core_thread * *dest*, struct btp_core_thread * *item*) [read]

Appends 'item' at the end of the list 'dest'.

**Returns:**

This function returns the 'dest' thread. If 'dest' is NULL, it returns the 'item' frame.

#### 5.6.2.2 void btp_core_thread_append_to_str (struct btp_core_thread * *thread*, struct btp_strbuf * *dest*)

Appends a textual representation of a thread to a string buffer.

#### 5.6.2.3 int btp_core_thread_cmp (struct btp_core_thread * *thread1*, struct btp_core_thread * *thread2*)

Compares two threads. When comparing the threads, it compares also their frames, including the frame numbers.

**Returns:**

Returns 0 if the threads are same. Returns negative number if t1 is found to be 'less' than t2. Returns positive number if t1 is found to be 'greater' than t2.

#### 5.6.2.4 struct btp_core_thread* btp_core_thread_dup (struct btp_core_thread * *thread*, bool *siblings*) [read]

Creates a duplicate of the thread.

**Parameters:**

***thread*** It must be non-NULL pointer. The thread is not modified by calling this function.

***siblings*** Whether to duplicate also siblings referenced by thread->next. If false, thread->next is not duplicated for the new frame, but it is set to NULL.

#### 5.6.2.5 void btp_core_thread_free (struct btp_core_thread * *thread*)

Releases the memory held by the thread. The thread siblings are not released.

**Parameters:**

***thread*** If thread is NULL, no operation is performed.

#### 5.6.2.6 int btp_core_thread_get_frame_count (struct btp_core_thread * *thread*)

Returns the number of frames in the thread.

#### 5.6.2.7 void btp_core_thread_init (struct btp_core_thread * *thread*)

Initializes all members of the thread to default values. No memory is released, members are simply overwritten. This is useful for initializing a thread structure placed on the stack.

#### 5.6.2.8 struct btp_core_thread* btp_core_thread_new () [read]

Creates and initializes a new frame structure.

**Returns:**

It never returns NULL. The returned pointer must be released by calling the function btp_core_thread_-free().

## 5.7 disassembler.h File Reference

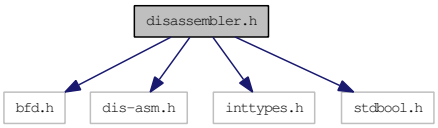
BFD-based function disassembler. `#include <bfd.h>`

`#include <dis-asm.h>`

`#include <inttypes.h>`

`#include <stdbool.h>`

Include dependency graph for disassembler.h:




### Data Structures

- struct btp_disasm_state

### Functions

- struct btp_disasm_state * **btp_disasm_init** (const char *file_name, char **error_message)
- void **btp_disasm_free** (struct btp_disasm_state *state)
- char ** btp_disasm_get_function_instructions (struct btp_disasm_state *state, uint64_t start_offset, uint64_t size, char **error_message)
- void **btp_disasm_function_instructions_free** (char **instructions)
- bool **btp_disasm_function_instruction_is_one_of** (const char *instruction, const char **mnemonics)
- bool **btp_disasm_function_instruction_present** (const char **instructions, const char **mnemonics)
- bool **btp_disasm_instruction_parse_single_address_operand** (const char *instruction, uint64_t *dest)
- uint64_t * **btp_disasm_get_callee_addresses** (const char **instructions)

### 5.7.1 Detailed Description

BFD-based function disassembler.

### 5.7.2 Function Documentation

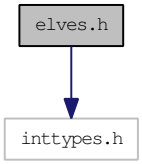
#### 5.7.2.1 char** **btp_disasm_get_function_instructions** (**struct btp_disasm_state** * *state*, **uint64_t** *start_offset*, **uint64_t** *size*, **char** ** *error_message*)

Disassemble the function starting at 'start_offset' and taking 'size' bytes, returning a list of (char*) instructions.

## 5.8 elves.h File Reference

Loading PLT and FDEs from ELF binaries. `#include <inttypes.h>`

Include dependency graph for elves.h:




### Data Structures

- struct btp_elf_plt_entry
- struct btp_elf_frame_description_entry

### Functions

- struct btp_elf_plt_entry * btp_elf_get_procedure_linkage_table (const char *filename, char **error_message)
- void **btp_elf_procedure_linkage_table_free** (struct btp_elf_plt_entry *entries)
- struct btp_elf_frame_description_entry * btp_elf_get_eh_frame (const char *filename, char **error_message)
- void **btp_elf_eh_frame_free** (struct btp_elf_frame_description_entry *entries)
- struct btp_elf_frame_description_entry * **btp_elf_find_fde_for_address** (struct btp_elf_frame_description_entry *eh_frame, uint64_t build_id_offset)

### 5.8.1 Detailed Description

Loading PLT and FDEs from ELF binaries.

### 5.8.2 Function Documentation

#### 5.8.2.1 struct btp_elf_frame_description_entry* btp_elf_get_eh_frame (const char * *filename*, char ** *error_message*) [read]

Reads the .eh_frame section from an ELF file.

**Parameters:**

***error_message*** Will be filled by an error message if the function fails (returns NULL). Caller is responsible for calling free() on the string pointer. If function succeeds, the pointer is not touched by the function.

**Returns:**

Returns a linked list of function ranges (function offset and size) on success. Otherwise NULL.

#### 5.8.2.2 struct btp_elf_plt_entry* btp_elf_get_procedure_linkage_table (const char * *filename*, char ** *error_message*) [read]

Reads the Procedure Linkage Table from an ELF file.

**Parameters:**

***error_message*** Will be filled by an error message if the function fails (returns NULL). Caller is responsible for calling free() on the string pointer. If function succeeds, the pointer is not touched by the function.

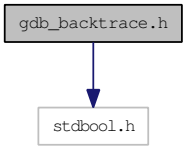
**Returns:**

Linked list of PLT entries on success. NULL otherwise.

## 5.9 gdb_backtrace.h File Reference

Stack trace as produced by GDB. `#include <stdbool.h>`

Include dependency graph for gdb_backtrace.h:




### Data Structures

- struct btp_gdb_backtrace

### Functions

- struct btp_gdb_backtrace * btp_gdb_backtrace_new ()
- void btp_gdb_backtrace_init (struct btp_gdb_backtrace *backtrace)
- void btp_gdb_backtrace_free (struct btp_gdb_backtrace *backtrace)
- struct btp_gdb_backtrace * btp_gdb_backtrace_dup (struct btp_gdb_backtrace *backtrace)
- int btp_gdb_backtrace_get_thread_count (struct btp_gdb_backtrace *backtrace)
- void btp_gdb_backtrace_remove_threads_except_one (struct btp_gdb_backtrace *backtrace, struct btp_gdb_thread *thread)
- struct btp_gdb_thread * btp_gdb_backtrace_find_crash_thread (struct btp_gdb_backtrace *backtrace)
- void btp_gdb_backtrace_limit_frame_depth (struct btp_gdb_backtrace *backtrace, int depth)
- float btp_gdb_backtrace_quality_simple (struct btp_gdb_backtrace *backtrace)
- float btp_gdb_backtrace_quality_complex (struct btp_gdb_backtrace *backtrace)
- char * btp_gdb_backtrace_to_text (struct btp_gdb_backtrace *backtrace, bool verbose)
- struct btp_gdb_frame * btp_gdb_backtrace_get_crash_frame (struct btp_gdb_backtrace *backtrace)
- char * btp_gdb_backtrace_get_duplication_hash (struct btp_gdb_backtrace *backtrace)
- struct btp_gdb_backtrace * btp_gdb_backtrace_parse (const char **input, struct btp_location *location)
- bool btp_gdb_backtrace_parse_header (const char **input, struct btp_gdb_frame **frame, struct btp_location *location)
- void btp_gdb_backtrace_set_libnames (struct btp_gdb_backtrace *backtrace)
- struct btp_gdb_thread * btp_gdb_backtrace_get_optimized_thread (struct btp_gdb_backtrace *backtrace, int max_frames)

### 5.9.1 Detailed Description

Stack trace as produced by GDB.

### 5.9.2 Function Documentation

#### 5.9.2.1 struct btp_gdb_backtrace* btp_gdb_backtrace_dup (struct btp_gdb_backtrace * *backtrace*) [read]

Creates a duplicate of the backtrace.

**Parameters:**

***backtrace*** The backtrace to be copied. It's not modified by this function.

**Returns:**

This function never returns NULL. The returned duplicate must be released by calling the function btp_gdb_backtrace_free().

#### 5.9.2.2 struct btp_gdb_thread* btp_gdb_backtrace_find_crash_thread (struct btp_gdb_backtrace * *backtrace*) [read]

Searches all threads and tries to find the one that caused the crash. It might return NULL if the thread cannot be determined.

**Parameters:**

***backtrace*** It must be non-NULL pointer. It's not modified by calling this function.

#### 5.9.2.3 void btp_gdb_backtrace_free (struct btp_gdb_backtrace * *backtrace*)

Releases the memory held by the backtrace, its threads, frames, shared libraries.

**Parameters:**

***backtrace*** If the backtrace is NULL, no operation is performed.

#### 5.9.2.4 struct btp_gdb_frame* btp_gdb_backtrace_get_crash_frame (struct btp_gdb_backtrace * *backtrace*) [read]

Analyzes the backtrace to get the frame where a crash occurred.

**Parameters:**

***backtrace*** It must be non-NULL pointer. It's not modified by calling this function.

**Returns:**

The returned value must be released by calling btp_gdb_frame_free() when it's no longer needed, because it is a deep copy of the crash frame from the backtrace. NULL is returned if the crash frame is not found.

#### 5.9.2.5 char* btp_gdb_backtrace_get_duplication_hash (struct btp_gdb_backtrace * *backtrace*)

Calculates the duplication hash string of the backtrace.

**Parameters:**

***backtrace*** It must be non-NULL pointer. It's not modified by calling this function.

**Returns:**

This function never returns NULL. The caller is responsible for releasing the returned memory using function free().

#### 5.9.2.6 struct btp_gdb_thread* btp_gdb_backtrace_get_optimized_thread (struct btp_gdb_backtrace * *backtrace*, int *max_frames*) [read]

Return crash thread optimized for comparison. It's normalized, with library names set and functions without names (signal handlers) are removed.

**Parameters:**

***backtrace*** It must be non-NULL pointer. It's not modified by calling this function.

***max_frames*** The maximum number of frames in the returned crash thread. Superfluous frames are removed from the returned thread.

**Returns:**

A newly allocated thread structure or NULL. NULL is returned when the crashing thread could not be found. The returned structure should be released by btp_gdb_thread_free() by the caller.

#### 5.9.2.7 int btp_gdb_backtrace_get_thread_count (struct btp_gdb_backtrace * *backtrace*)

Returns a number of threads in the backtrace.

**Parameters:**

***backtrace*** It's not modified by calling this function.

#### 5.9.2.8 void btp_gdb_backtrace_init (struct btp_gdb_backtrace * *backtrace*)

Initializes all members of the backtrace structure to their default values. No memory is released, members are simply overwritten. This is useful for initializing a backtrace structure placed on the stack.

#### 5.9.2.9 void btp_gdb_backtrace_limit_frame_depth (struct btp_gdb_backtrace * *backtrace*, int *depth*)

Remove frames from the bottom of threads in the backtrace, until all threads have at most 'depth' frames.

**Parameters:**

***backtrace*** Must be non-NULL pointer.

#### 5.9.2.10 struct btp_gdb_backtrace* btp_gdb_backtrace_new () [read]

Creates and initializes a new backtrace structure.

**Returns:**

It never returns NULL. The returned pointer must be released by calling the function btp_gdb_-backtrace_free().

#### 5.9.2.11 struct btp_gdb_backtrace* btp_gdb_backtrace_parse (const char ** *input*, struct btp_location * *location*) [read]

Parses a textual backtrace and puts it into a structure. If parsing fails, the input parameter is not changed and NULL is returned.

```
struct btp_location location;
btp_location_init(&location);
char *input = "...";
struct btp_gdb_backtrace *backtrace;
backtrace = btp_gdb_backtrace_parse(input, location);
if (!backtrace)
{
  fprintf(stderr,
          "Failed to parse the backtrace.\n"
          "Line %d, column %d: %s\n",
          location.line,
          location.column,
          location.message);
  exit(-1);
}
btp_gdb_backtrace_free(backtrace);
```

**Parameters:**

***input*** Pointer to the string with the backtrace. If this function returns true, this pointer is modified to point after the backtrace that was just parsed.

***location*** The caller must provide a pointer to an instance of btp_location here. The line and column members of the location are gradually increased as the parser handles the input, so the location should be initialized by btp_location_init() before calling this function to get reasonable values. When this function returns false (an error occurred), the structure will contain the error line, column, and message.

**Returns:**

A newly allocated backtrace structure or NULL. A backtrace struct is returned when at least one thread was parsed from the input and no error occurred. The returned structure should be released by btp_-gdb_backtrace_free().

#### 5.9.2.12 bool btp_gdb_backtrace_parse_header (const char ** *input*, struct btp_gdb_frame ** *frame*, struct btp_location * *location*)

Parse backtrace header if it is available in the backtrace. The header usually contains frame where the program crashed.

**Parameters:**

***input*** Pointer that will be moved to point behind the header if the header is successfully detected and parsed.

***frame*** If this function succeeds and returns true, ∗frame contains the crash frame that is usually a part of the header. If no frame is detected in the header, ∗frame is set to NULL.

```
[New Thread 11919]
[New Thread 11917]
Core was generated by ‘evince file:
Program terminated with signal 8, Arithmetic exception.
#0  0x000000322a2362b9 in repeat (image=<value optimized out>,
    mask=<value optimized out>, mask_bits=<value optimized out>)
    at pixman-bits-image.c:145
145     pixman-bits-image.c: No such file or directory.
        in pixman-bits-image.c
```

#### 5.9.2.13 float btp_gdb_backtrace_quality_complex (struct btp_gdb_backtrace ∗ *backtrace*)

Evaluates the quality of the backtrace. The quality is determined depending on the ratio of frames with function name fully known to all frames.

**Parameters:**

***backtrace*** It must be non-NULL pointer. It's not modified by calling this function.

**Returns:**

A number between 0 and 1. 0 means the lowest quality, 1 means full backtrace is known. The returned value takes into account that the thread which caused the crash is more important than the other threads, and the frames around the crash frame are more important than distant frames.

#### 5.9.2.14 float btp_gdb_backtrace_quality_simple (struct btp_gdb_backtrace ∗ *backtrace*)

Evaluates the quality of the backtrace. The quality is the ratio of the number of frames with function name fully known to the number of all frames. This function does not take into account that some frames are more important than others.

**Parameters:**

***backtrace*** It must be non-NULL pointer. It's not modified by calling this function.

**Returns:**

A number between 0 and 1. 0 means the lowest quality, 1 means full backtrace is known (all function names are known).

#### 5.9.2.15 void btp_gdb_backtrace_remove_threads_except_one (struct btp_gdb_backtrace ∗ *backtrace*, struct btp_gdb_thread ∗ *thread*)

Removes all threads from the backtrace and deletes them, except the one provided as a parameter.

**Parameters:**

***thread*** This function does not check whether the thread is a member of the backtrace. If it's not, all threads are removed from the backtrace and then deleted.

#### 5.9.2.16 void btp_gdb_backtrace_set_libnames (struct btp_gdb_backtrace * *backtrace*)

Set library names in all frames in the backtrace according to the the sharedlib data.

#### 5.9.2.17 char* btp_gdb_backtrace_to_text (struct btp_gdb_backtrace * *backtrace*, bool *verbose*)

Returns textual representation of the backtrace.

**Parameters:**

***backtrace*** It must be non-NULL pointer. It's not modified by calling this function.

**Returns:**

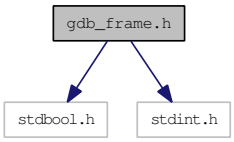
This function never returns NULL. The caller is responsible for releasing the returned memory using function free().

## 5.10 gdb_frame.h File Reference

Single frame of GDB stack trace thread. `#include <stdbool.h>`

`#include <stdint.h>`

Include dependency graph for gdb_frame.h:




### Data Structures

- struct btp_gdb_frame

### Functions

- struct btp_gdb_frame * btp_gdb_frame_new ()
- void btp_gdb_frame_init (struct btp_gdb_frame *frame)
- void btp_gdb_frame_free (struct btp_gdb_frame *frame)
- struct btp_gdb_frame * btp_gdb_frame_dup (struct btp_gdb_frame *frame, bool siblings)
- bool btp_gdb_frame_calls_func (struct btp_gdb_frame *frame, const char *function_name)
- bool btp_gdb_frame_calls_func_in_file (struct btp_gdb_frame *frame, const char *function_name, const char *source_file)
- bool btp_gdb_frame_calls_func_in_file2 (struct btp_gdb_frame *frame, const char *function_name, const char *source_file0, const char *source_file1)
- bool btp_gdb_frame_calls_func_in_file3 (struct btp_gdb_frame *frame, const char *function_name, const char *source_file0, const char *source_file1, const char *source_file2)
- bool btp_gdb_frame_calls_func_in_file4 (struct btp_gdb_frame *frame, const char *function_name, const char *source_file0, const char *source_file1, const char *source_file2, const char *source_file3)
- int btp_gdb_frame_cmp (struct btp_gdb_frame *frame1, struct btp_gdb_frame *frame2, bool compare_number)
- int btp_gdb_frame_cmp_simple (struct btp_gdb_frame *frame1, struct btp_gdb_frame *frame2)
- struct btp_gdb_frame * btp_gdb_frame_append (struct btp_gdb_frame *dest, struct btp_gdb_frame *item)
- void btp_gdb_frame_append_to_str (struct btp_gdb_frame *frame, struct btp_strbuf *dest, bool verbose)
- struct btp_gdb_frame * btp_gdb_frame_parse (const char **input, struct btp_location *location)
- int btp_gdb_frame_parse_frame_start (const char **input, unsigned *number)
- int btp_gdb_frame_parseadd_operator (const char **input, struct btp_strbuf *target)
- int btp_gdb_frame_parse_function_name_chunk (const char **input, bool space_allowed, char **target)
- int btp_gdb_frame_parse_function_name_braces (const char **input, char **target)
- int btp_gdb_frame_parse_function_name_template (const char **input, char **target)
- bool btp_gdb_frame_parse_function_name (const char **input, char **function_name, char **function_type, struct btp_location *location)
- bool btp_gdb_frame_skip_function_args (const char **input, struct btp_location *location)

- bool btp_gdb_frame_parse_function_call (const char **input, char **function_name, char **function_type, struct btp_location *location)
- bool btp_gdb_frame_parse_address_in_function (const char **input, uint64_t *address, char **function_name, char **function_type, struct btp_location *location)
- bool btp_gdb_frame_parse_file_location (const char **input, char **file, unsigned *fileline, struct btp_location *location)
- struct btp_gdb_frame * btp_gdb_frame_parse_header (const char **input, struct btp_location *location)
- void btp_gdb_frame_remove_func_prefix (struct btp_gdb_frame *frame, const char *prefix, int num)

### 5.10.1 Detailed Description

Single frame of GDB stack trace thread.

### 5.10.2 Function Documentation

#### 5.10.2.1 struct btp_gdb_frame* btp_gdb_frame_append (struct btp_gdb_frame * *dest*, struct btp_gdb_frame * *item*) [read]

Appends 'item' at the end of the list 'dest'.

**Returns:**

This function returns the 'dest' frame. If 'dest' is NULL, it returns the 'item' frame.

#### 5.10.2.2 void btp_gdb_frame_append_to_str (struct btp_gdb_frame * *frame*, struct btp_strbuf * *dest*, bool *verbose*)

Appends the textual representation of the frame to the string buffer.

**Parameters:**

***frame*** It must be a non-NULL pointer. It's not modified by calling this function.

#### 5.10.2.3 bool btp_gdb_frame_calls_func (struct btp_gdb_frame * *frame*, const char * *function_name*)

Checks whether the frame represents a call of function with certain function name.

#### 5.10.2.4 bool btp_gdb_frame_calls_func_in_file (struct btp_gdb_frame * *frame*, const char * *function_name*, const char * *source_file*)

Checks whether the frame represents a call of function with certain function name, which resides in a source file.

**Parameters:**

***source_file*** The frame's source_file is searched for the source_file as a substring.

#### 5.10.2.5 bool btp_gdb_frame_calls_func_in_file2 (struct btp_gdb_frame * *frame*, const char * *function_name*, const char * *source_file0*, const char * *source_file1*)

Checks whether the frame represents a call of function with certain function name, which resides in one of the source files.

**Parameters:**

***source_file0*** The frame's source_file is searched for the source_file0 as a substring.

**Returns:**

True if the frame corresponds to a function with function_name, residing in the source_file0, or source_file1.

#### 5.10.2.6 bool btp_gdb_frame_calls_func_in_file3 (struct btp_gdb_frame * *frame*, const char * *function_name*, const char * *source_file0*, const char * *source_file1*, const char * *source_file2*)

Checks whether the frame represents a call of function with certain function name, which resides in one of the source files.

**Parameters:**

***source_file0*** The frame's source_file is searched for the source_file0 as a substring.

**Returns:**

True if the frame corresponds to a function with function_name, residing in the source_file0, source_-file1, or source_file2.

#### 5.10.2.7 bool btp_gdb_frame_calls_func_in_file4 (struct btp_gdb_frame * *frame*, const char * *function_name*, const char * *source_file0*, const char * *source_file1*, const char * *source_file2*, const char * *source_file3*)

Checks whether the frame represents a call of function with certain function name, which resides in one of the source files.

**Parameters:**

***source_file0*** The frame's source_file is searched for the source_file0 as a substring.

**Returns:**

True if the frame corresponds to a function with function_name, residing in the source_file0, source_-file1, source_file2, or source_file3.

#### 5.10.2.8 int btp_gdb_frame_cmp (struct btp_gdb_frame * *frame1*, struct btp_gdb_frame * *frame2*, bool *compare_number*)

Compares two frames.

**Parameters:**

***frame1*** It must be non-NULL pointer. It's not modified by calling this function.

***frame2*** It must be non-NULL pointer. It's not modified by calling this function.

***compare_number*** Indicates whether to include the frame numbers in the comparsion. If set to false, the frame numbers are ignored.

**Returns:**

Returns 0 if the frames are same. Returns negative number if frame1 is found to be 'less' than frame2. Returns positive number if frame1 is found to be 'greater' than frame2.

#### 5.10.2.9 int btp_gdb_frame_cmp_simple (struct btp_gdb_frame * *frame1*, struct btp_gdb_frame * *frame2*)

Compares two frames, but only by their function and library names. Two unknown functions ("??") are assumed to be different and unknown library names to be the same.

**Parameters:**

***frame1*** It must be non-NULL pointer. It's not modified by calling this function.

***frame2*** It must be non-NULL pointer. It's not modified by calling this function.

**Returns:**

Returns 0 if the frames are same. Returns negative number if frame1 is found to be 'less' than frame2. Returns positive number if frame1 is found to be 'greater' than frame2.

#### 5.10.2.10 struct btp_gdb_frame* btp_gdb_frame_dup (struct btp_gdb_frame * *frame*, bool *siblings*) [read]

Creates a duplicate of the frame.

**Parameters:**

***frame*** It must be non-NULL pointer. The frame is not modified by calling this function.

***siblings*** Whether to duplicate also siblings referenced by frame->next. If false, frame->next is not duplicated for the new frame, but it is set to NULL.

**Returns:**

This function never returns NULL. If the returned duplicate is not shallow, it must be released by calling the function btp_gdb_frame_free().

#### 5.10.2.11 void btp_gdb_frame_free (struct btp_gdb_frame * *frame*)

Releases the memory held by the frame. The frame siblings are not released.

**Parameters:**

***frame*** If the frame is NULL, no operation is performed.

#### 5.10.2.12 void btp_gdb_frame_init (struct btp_gdb_frame * *frame*)

Initializes all members of the frame structure to their default values. No memory is released, members are simply overwritten. This is useful for initializing a frame structure placed on the stack.

#### 5.10.2.13 struct btp_gdb_frame* btp_gdb_frame_new () [read]

Creates and initializes a new frame structure.

**Returns:**

It never returns NULL. The returned pointer must be released by calling the function btp_gdb_frame_-free().

#### 5.10.2.14 struct btp_gdb_frame* btp_gdb_frame_parse (const char ** *input*, struct btp_location * *location*) [read]

If the input contains a complete frame, this function parses the frame text, returns it in a structure, and moves the input pointer after the frame. If the input does not contain proper, complete frame, the function does not modify input and returns NULL.

**Returns:**

Allocated pointer with a frame structure. The pointer should be released by btp_gdb_frame_free().

**Parameters:**

***location*** The caller must provide a pointer to an instance of btp_location here. When this function returns NULL, the structure will contain the error line, column, and message. The line and column members of the location are gradually increased as the parser handles the input, so the location should be initialized before calling this function to get reasonable values.

#### 5.10.2.15 bool btp_gdb_frame_parse_address_in_function (const char ** *input*, uint64_t * *address*, char ** *function_name*, char ** *function_type*, struct btp_location * *location*)

If the input contains address and function call, parse them, move the input pointer after this sequence, and return true. Otherwise do not modify the input and return false.

If this function returns true, the caller is responsible to free the parameter function.

```
0x000000322160e7fd in fsync ()
0x000000322222987a in write_to_temp_file (
filename=0x18971b0 "/home/jfclere/.recently-used.xbel",
contents=<value optimized out>, length=29917, error=0x7fff3cbe4110)
```

**Parameters:**

***location*** The caller must provide a pointer to an instance of btp_location here. The line and column members of the location are gradually increased as the parser handles the input, so the location should be initialized before calling this function to get reasonable values. When this function returns false (an error occurred), the structure will contain the error line, column, and message.

#### 5.10.2.16 bool btp_gdb_frame_parse_file_location (const char ** *input*, char ** *file*, unsigned * *fileline*, struct btp_location * *location*)

If the input contains sequence "from path/to/file:fileline" or "at path/to/file:fileline", parse it, move the input pointer after this sequence and return true. Otherwise do not modify the input and return false.

The ':' followed by line number is optional. If it is not present, the fileline is set to -1.

**Parameters:**

***location*** The caller must provide a pointer to an instance of btp_location here. The line and column members of the location are gradually increased as the parser handles the input, so the location should be initialized before calling this function to get reasonable values. When this function returns false (an error occurred), the structure will contain the error line, column, and message.

#### 5.10.2.17 int btp_gdb_frame_parse_frame_start (const char ** *input*, unsigned * *number*)

If the input contains a proper frame start section, parse the frame number, and move the input pointer after this section. Otherwise do not modify input.

**Returns:**

The number of characters parsed from input. 0 if the input does not contain a frame start.

```
"#1 "
"#255  "
```

#### 5.10.2.18 bool btp_gdb_frame_parse_function_call (const char ** *input*, char ** *function_name*, char ** *function_type*, struct btp_location * *location*)

If the input contains proper function call, parse the function name and store it to result, move the input pointer after whole function call, and return true. Otherwise do not modify the input and return false.

If this function returns true, the caller is responsible to free the the function_name.

**Parameters:**

***location*** The caller must provide a pointer to an instance of btp_location here. The line and column members of the location are gradually increased as the parser handles the input, so the location should be initialized before calling this function to get reasonable values. When this function returns false (an error occurred), the structure will contain the error line, column, and message.

#### 5.10.2.19 bool btp_gdb_frame_parse_function_name (const char ** *input*, char ** *function_name*, char ** *function_type*, struct btp_location * *location*)

Parses the function name, which is a part of the frame header, from the input. If the frame header contains also the function type, it's also parsed.

**Parameters:**

***function_name*** A pointer pointing to an uninitialized pointer. This function allocates a string and sets the pointer to it if it parses the function name from the input successfully. The memory returned this way must be released by the caller using the function free(). If this function returns true, this pointer is guaranteed to be non-NULL.

***location*** The caller must provide a pointer to an instance of btp_location here. The line and column members of the location are gradually increased as the parser handles the input, so the location should be initialized before calling this function to get reasonable values. When this function returns false (an error occurred), the structure will contain the error line, column, and message.

**Returns:**

True if the input stream contained a function name, which has been parsed. False otherwise.

#### 5.10.2.20 int btp_gdb_frame_parse_function_name_braces (const char ** *input*, char ** *target*)

If the input buffer contains part of function name containing braces, for example "(anonymous namespace)", parse it, append the contents to target and move input after the braces. Otherwise do not modify the input and the target.

**Returns:**

The number of characters parsed from input. 0 if the input does not contain a braced part of function name.

#### 5.10.2.21 int btp_gdb_frame_parse_function_name_chunk (const char ** *input*, bool *space_allowed*, char ** *target*)

Parses a part of function name from the input.

**Parameters:**

***target*** Pointer to a non-allocated pointer. This function will set the pointer to newly allocated memory containing the name chunk, if it returns positive, nonzero value.

**Returns:**

The number of characters parsed from input. 0 if the input does not contain a part of function name.

#### 5.10.2.22 int btp_gdb_frame_parse_function_name_template (const char ** *input*, char ** *target*)

**Returns:**

The number of characters parsed from input. 0 if the input does not contain a template part of function name.

#### 5.10.2.23 struct btp_gdb_frame* btp_gdb_frame_parse_header (const char ** *input*, struct btp_location * *location*) [read]

If the input contains proper frame header, this function parses the frame header text, moves the input pointer after the frame header, and returns a frame struct. If the input does not contain proper frame header, this function returns NULL and does not modify input.

**Parameters:**

***location*** The caller must provide a pointer to an instance of btp_location here. The line and column members of the location are gradually increased as the parser handles the input, so the location

should be initialized before calling this function to get reasonable values. When this function returns false (an error occurred), the structure will contain the error line, column, and message.

**Returns:**

Newly created frame struct or NULL. The returned frame struct should be released by btp_gdb_-frame_free().

#### 5.10.2.24 int btp_gdb_frame_parseadd_operator (const char ** *input*, struct btp_strbuf * *target*)

Parses C++ operator on input. Supports even 'operator new[]' and 'operator delete[]'.

**Parameters:**

***target*** The parsed operator name is appened to the string buffer provided, if an operator is found. Otherwise the string buffer is not changed.

**Returns:**

The number of characters parsed from input. 0 if the input does not contain operator.

#### 5.10.2.25 void btp_gdb_frame_remove_func_prefix (struct btp_gdb_frame * *frame*, const char * *prefix*, int *num*)

Removes first num chars from function name in the frame if it begins with the prefix.

#### 5.10.2.26 bool btp_gdb_frame_skip_function_args (const char ** *input*, struct btp_location * *location*)

Skips function arguments which are a part of the frame header, in the input stream.

**Parameters:**

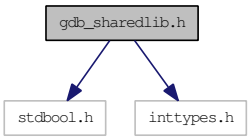
***location*** The caller must provide a pointer to an instance of btp_location here. The line and column members of the location are gradually increased as the parser handles the input, so the location should be initialized before calling this function to get reasonable values. When this function returns false (an error occurred), the structure will contain the error line, column, and message.

## 5.11 gdb_sharedlib.h File Reference

Shared library information as produced by GDB. `#include <stdbool.h>`

`#include <inttypes.h>`

Include dependency graph for gdb_sharedlib.h:




### Data Structures

- struct btp_gdb_sharedlib

### Enumerations

- enum { **SYMS_OK**, **SYMS_WRONG**, **SYMS_NOT_FOUND** }

### Functions

- struct btp_gdb_sharedlib * btp_gdb_sharedlib_new ()
- void btp_gdb_sharedlib_init (struct btp_gdb_sharedlib *sharedlib)
- void btp_gdb_sharedlib_free (struct btp_gdb_sharedlib *sharedlib)
- struct btp_gdb_sharedlib * btp_gdb_sharedlib_append (struct btp_gdb_sharedlib *dest, struct btp_gdb_sharedlib *item)
- struct btp_gdb_sharedlib * btp_gdb_sharedlib_dup (struct btp_gdb_sharedlib *sharedlib, bool siblings)
- int btp_gdb_sharedlib_count (struct btp_gdb_sharedlib *sharedlib)
- struct btp_gdb_sharedlib * btp_gdb_sharedlib_find_address (struct btp_gdb_sharedlib *first, uint64_t address)
- struct btp_gdb_sharedlib * btp_gdb_sharedlib_parse (const char *input)

### 5.11.1 Detailed Description

Shared library information as produced by GDB.

### 5.11.2 Function Documentation

#### 5.11.2.1 struct btp_gdb_sharedlib* btp_gdb_sharedlib_append (struct btp_gdb_sharedlib * *dest*, struct btp_gdb_sharedlib * *item*) `[read]`

Appends 'item' at the end of the list 'dest'.

**Returns:**

This function returns the 'dest' sharedlib. If 'dest' is NULL, it returns the 'item' sharedlib.

#### 5.11.2.2 int btp_gdb_sharedlib_count (struct btp_gdb_sharedlib * *sharedlib*)

Returns the number of sharedlibs in the list.

#### 5.11.2.3 struct btp_gdb_sharedlib* btp_gdb_sharedlib_dup (struct btp_gdb_sharedlib * *sharedlib*, bool *siblings*) [read]

Creates a duplicate of the sharedlib structure.

**Parameters:**

***sharedlib*** Structure to be duplicated.

***siblings*** Whether to duplicate a single structure or whole list.

**Returns:**

Never returns NULL. Returns the duplicated structure or the first structure in the duplicated list.

#### 5.11.2.4 struct btp_gdb_sharedlib* btp_gdb_sharedlib_find_address (struct btp_gdb_sharedlib * *first*, uint64_t *address*) [read]

Finds whether the address belongs to some sharedlib from the list starting by 'first'.

**Returns:**

Pointer to an existing structure or NULL if not found.

#### 5.11.2.5 void btp_gdb_sharedlib_free (struct btp_gdb_sharedlib * *sharedlib*)

Releases the memory held by the sharedlib. Sharedlibs referenced by .next are not released.

**Parameters:**

***sharedlib*** If sharedlib is NULL, no operation is performed.

#### 5.11.2.6 void btp_gdb_sharedlib_init (struct btp_gdb_sharedlib * *sharedlib*)

Initializes all members of the sharedlib to default values. No memory is released, members are simply overwritten. This is useful for initializing a sharedlib structure placed on the stack.

#### 5.11.2.7 struct btp_gdb_sharedlib* btp_gdb_sharedlib_new () [read]

Creates and initializes a new sharedlib structure.

**Returns:**

It never returns NULL. The returned pointer must be released by calling the function btp_gdb_-sharedlib_free().

#### 5.11.2.8 struct btp_gdb_sharedlib* btp_gdb_sharedlib_parse (const char * *input*) [read]

Parses the output of GDB's 'info sharedlib' command.

**Parameters:**

***input*** String representing the backtrace.

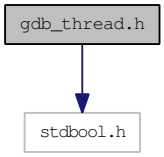
**Returns:**

First element of the list of loaded libraries.

## 5.12 gdb_thread.h File Reference

Single thread of execution of GDB stack trace. `#include <stdbool.h>`

Include dependency graph for gdb_thread.h:




### Data Structures

- struct btp_gdb_thread

### Functions

- struct btp_gdb_thread ∗ btp_gdb_thread_new ()
- void btp_gdb_thread_init (struct btp_gdb_thread ∗thread)
- void btp_gdb_thread_free (struct btp_gdb_thread ∗thread)
- struct btp_gdb_thread ∗ btp_gdb_thread_dup (struct btp_gdb_thread ∗thread, bool siblings)
- int btp_gdb_thread_cmp (struct btp_gdb_thread ∗thread1, struct btp_gdb_thread ∗thread2)
- struct btp_gdb_thread ∗ btp_gdb_thread_append (struct btp_gdb_thread ∗dest, struct btp_gdb_thread ∗item)
- int btp_gdb_thread_get_frame_count (struct btp_gdb_thread ∗thread)
- void btp_gdb_thread_quality_counts (struct btp_gdb_thread ∗thread, int ∗ok_count, int ∗all_count)
- float btp_gdb_thread_quality (struct btp_gdb_thread ∗thread)
- bool btp_gdb_thread_remove_frame (struct btp_gdb_thread ∗thread, struct btp_gdb_frame ∗frame)
- bool btp_gdb_thread_remove_frames_above (struct btp_gdb_thread ∗thread, struct btp_gdb_frame ∗frame)
- void btp_gdb_thread_remove_frames_below_n (struct btp_gdb_thread ∗thread, int n)
- void btp_gdb_thread_append_to_str (struct btp_gdb_thread ∗thread, struct btp_strbuf ∗dest, bool verbose)
- struct btp_gdb_thread ∗ btp_gdb_thread_parse (const char ∗∗input, struct btp_location ∗location)
- int btp_gdb_thread_skip_lwp (const char ∗∗input)
- struct btp_gdb_thread ∗ btp_gdb_thread_parse_funs (const char ∗input)
- char ∗ btp_gdb_thread_format_funs (struct btp_gdb_thread ∗thread)

### 5.12.1 Detailed Description

Single thread of execution of GDB stack trace.

### 5.12.2 Function Documentation

#### 5.12.2.1 struct btp_gdb_thread∗ btp_gdb_thread_append (struct btp_gdb_thread ∗ *dest*, struct btp_gdb_thread ∗ *item*) [read]

Appends 'item' at the end of the list 'dest'.

**Returns:**

This function returns the 'dest' thread.

#### 5.12.2.2 void btp_gdb_thread_append_to_str (struct btp_gdb_thread * *thread*, struct btp_strbuf * *dest*, bool *verbose*)

Appends a textual representation of 'thread' to the 'str'.

#### 5.12.2.3 int btp_gdb_thread_cmp (struct btp_gdb_thread * *thread1*, struct btp_gdb_thread * *thread2*)

Compares two threads. When comparing the threads, it compares also their frames, including the frame numbers.

**Returns:**

Returns 0 if the threads are same. Returns negative number if t1 is found to be 'less' than t2. Returns positive number if t1 is found to be 'greater' than t2.

#### 5.12.2.4 struct btp_gdb_thread* btp_gdb_thread_dup (struct btp_gdb_thread * *thread*, bool *siblings*) [read]

Creates a duplicate of the thread.

**Parameters:**

***thread*** It must be non-NULL pointer. The thread is not modified by calling this function.

***siblings*** Whether to duplicate also siblings referenced by thread->next. If false, thread->next is not duplicated for the new frame, but it is set to NULL.

#### 5.12.2.5 char* btp_gdb_thread_format_funs (struct btp_gdb_thread * *thread*)

Prepare a string representing thread which contains just the function and library names. This can be used to store only data necessary for comparison.

**Returns:**

Newly allocated string, which should be released by calling free(). The string can be parsed by btp_gdb_thread_parse_funs().

#### 5.12.2.6 void btp_gdb_thread_free (struct btp_gdb_thread * *thread*)

Releases the memory held by the thread. The thread siblings are not released.

**Parameters:**

***thread*** If thread is NULL, no operation is performed.

#### 5.12.2.7 int btp_gdb_thread_get_frame_count (struct btp_gdb_thread * *thread*)

Returns the number of frames in the thread.

#### 5.12.2.8 void btp_gdb_thread_init (struct btp_gdb_thread * *thread*)

Initializes all members of the thread to default values. No memory is released, members are simply overwritten. This is useful for initializing a thread structure placed on the stack.

#### 5.12.2.9 struct btp_gdb_thread* btp_gdb_thread_new () [read]

Creates and initializes a new frame structure.

**Returns:**

It never returns NULL. The returned pointer must be released by calling the function btp_gdb_thread_-free().

#### 5.12.2.10 struct btp_gdb_thread* btp_gdb_thread_parse (const char ** *input*, struct btp_location * *location*) [read]

If the input contains proper thread with frames, parse the thread, move the input pointer after the thread, and return a structure representing the thread. Otherwise to not modify the input pointer and return NULL.

**Parameters:**

***location*** The caller must provide a pointer to struct btp_location here. The line and column members are gradually increased as the parser handles the input, keep this in mind to get reasonable values. When this function returns NULL (an error occurred), the structure will contain the error line, column, and message.

**Returns:**

NULL or newly allocated structure, which should be released by calling btp_gdb_thread_free().

#### 5.12.2.11 struct btp_gdb_thread* btp_gdb_thread_parse_funs (const char * *input*) [read]

Create a thread from function and library names.

**Parameters:**

***input*** String containing function names and library names separated by space, one frame per line.

**Returns:**

Newly allocated structure, which should be released by calling btp_gdb_thread_free().

#### 5.12.2.12 float btp_gdb_thread_quality (struct btp_gdb_thread * *thread*)

Returns the quality of the thread. The quality is the ratio of the number of frames with function name fully known to the number of all frames. This function does not take into account that some frames are more important than others.

**Parameters:**

***thread*** Must be a non-NULL pointer. It's not modified in this function.

**Returns:**

A number between 0 and 1. 0 means the lowest quality, 1 means full thread backtrace is known. If the thread contains no frames, this function returns 1.

#### 5.12.2.13 void btp_gdb_thread_quality_counts (struct btp_gdb_thread * *thread*, int * *ok_count*, int * *all_count*)

Counts the number of 'good' frames and the number of all frames in a thread. Good means that the function name is known (so it's not just '??').

**Parameters:**

***ok_count***

***all_count*** Not zeroed. This function just adds the numbers to ok_count and all_count.

#### 5.12.2.14 bool btp_gdb_thread_remove_frame (struct btp_gdb_thread * *thread*, struct btp_gdb_frame * *frame*)

Removes the frame from the thread and then deletes it.

**Returns:**

True if the frame was found in the thread and removed and deleted. False if the frame was not found in the thread.

#### 5.12.2.15 bool btp_gdb_thread_remove_frames_above (struct btp_gdb_thread * *thread*, struct btp_gdb_frame * *frame*)

Removes all the frames from the thread that are above certain frame.

**Returns:**

True if the frame was found, and all the frames that were above the frame in the thread were removed from the thread and then deleted. False if the frame was not found in the thread.

#### 5.12.2.16 void btp_gdb_thread_remove_frames_below_n (struct btp_gdb_thread * *thread*, int *n*)

Keeps only the top n frames in the thread.

#### 5.12.2.17 int btp_gdb_thread_skip_lwp (const char ** *input*)

If the input contains a LWP section in form of (LWP [0-9]+), move the input pointer after this section. Otherwise do not modify input.

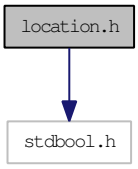
**Returns:**

The number of characters parsed from input. 0 if the input does not contain a LWP section.

## 5.13 location.h File Reference

Parser location in input file. `#include <stdbool.h>`

Include dependency graph for location.h:




### Data Structures

- struct btp_location

### Functions

- void btp_location_init (struct btp_location *location)
- int btp_location_cmp (struct btp_location *location1, struct btp_location *location2, bool compare_-messages)
- char * btp_location_to_string (struct btp_location *location)
- void btp_location_add (struct btp_location *location, int add_line, int add_column)
- void btp_location_add_ext (int *line, int *column, int add_line, int add_column)
- void btp_location_eat_char (struct btp_location *location, char c)
- void btp_location_eat_char_ext (int *line, int *column, char c)

### 5.13.1 Detailed Description

Parser location in input file.

### 5.13.2 Function Documentation

#### 5.13.2.1 void btp_location_add (struct btp_location * *location*, int *add_line*, int *add_column*)

Adds a line and a column to specific location.

**Note:**

If the line is not 1 (meaning the first line), the column in the location structure is overwritten by the provided add_column value. Otherwise the add_column value is added to the column member of the location structure.

**Parameters:**

***location*** The structure to be modified. It must be a valid pointer.

***add_line*** Starts from 1. It means that if add_line is 1, the line member of the location structure is not changed.

***add_column*** Starts from 0.

#### 5.13.2.2 void btp_location_add_ext (int * *line*, int * *column*, int *add_line*, int *add_column*)

Adds a line column pair to another line column pair.

**Note:**

If the add_line is not 1 (meaning the frist line), the column is overwritten by the provided add_column value. Otherwise the add_column value is added to the column.

**Parameters:**

***add_line*** Starts from 1. It means that if add_line is 1, the line is not changed.

***add_column*** Starts from 0.

#### 5.13.2.3 int btp_location_cmp (struct btp_location * *location1*, struct btp_location * *location2*, bool *compare_messages*)

Compare two locations.

**Parameters:**

***location1*** It must be non-NULL pointer. It's not modified by calling this function.

***location2*** It must be non-NULL pointer. It's not modified by calling this function.

***compare_messages*** Indicates whether to compare messages in the locations as well.

**Returns:**

Returns 0 if the locations are same. Returns negative number if location1 is found to be 'less' than location2. Returns positive number if location1 is found to be 'greater' than location2.

'Less' and 'greater' take lines into account first. If a location1 line is lower than location2 line, location1 is considered 'less' than location2. If the lines are the same, columns are compared. When compare_-messages is true and lines and columns are equal, the locations' messages are compared according to the lexicographical order.

#### 5.13.2.4 void btp_location_eat_char (struct btp_location * *location*, char *c*)

Updates the line and column of the location by moving "after" the char c. If c is a newline character, the line number is increased and the column is set to 0. Otherwise the column is increased by 1.

#### 5.13.2.5 void btp_location_eat_char_ext (int * *line*, int * *column*, char *c*)

Updates the line and the column by moving "after" the char c. If c is a newline character, the line number is increased and the column is set to 0. Otherwise the column is increased.

**Parameters:**

***line*** Must be a valid pointer.

***column*** Must be a valid pointer.

#### 5.13.2.6 void btp_location_init (struct btp_location * *location*)

Initializes all members of the location struct to their default values. No memory is allocated or released by this function.

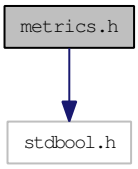
#### 5.13.2.7 char* btp_location_to_string (struct btp_location * *location*)

Creates a string representation of location. User must delete the returned string using free().

## 5.14 metrics.h File Reference

Distance between stack trace threads. `#include <stdbool.h>`

Include dependency graph for metrics.h:




### Data Structures

- struct btp_distances

### Typedefs

- typedef int(∗ **btp_gdb_frame_cmp_type** )(struct btp_gdb_frame ∗, struct btp_gdb_frame ∗)
- typedef float(∗ btp_dist_thread_type )(struct btp_gdb_thread ∗, struct btp_gdb_thread ∗)

### Functions

- float **btp_gdb_thread_jarowinkler_distance** (struct btp_gdb_thread ∗thread1, struct btp_gdb_-thread ∗thread2)
- float **btp_gdb_thread_jaccard_distance** (struct btp_gdb_thread ∗thread1, struct btp_gdb_thread ∗thread2)
- int **btp_gdb_thread_levenshtein_distance** (struct btp_gdb_thread ∗thread1, struct btp_gdb_thread ∗thread2, bool transposition)
- float **btp_gdb_thread_levenshtein_distance_f** (struct btp_gdb_thread ∗thread1, struct btp_gdb_-thread ∗thread2)
- float **btp_gdb_thread_jarowinkler_distance_custom** (struct btp_gdb_thread ∗thread1, struct btp_-gdb_thread ∗thread2, btp_gdb_frame_cmp_type compare_func)
- float **btp_gdb_thread_jaccard_distance_custom** (struct btp_gdb_thread ∗thread1, struct btp_-gdb_thread ∗thread2, btp_gdb_frame_cmp_type compare_func)
- int **btp_gdb_thread_levenshtein_distance_custom** (struct btp_gdb_thread ∗thread1, struct btp_-gdb_thread ∗thread2, bool transposition, btp_gdb_frame_cmp_type compare_func)
- struct btp_distances ∗ btp_distances_new (int m, int n)
- struct btp_distances ∗ btp_distances_dup (struct btp_distances ∗distances)
- void btp_distances_free (struct btp_distances ∗distances)
- float btp_distances_get_distance (struct btp_distances ∗distances, int i, int j)
- void btp_distances_set_distance (struct btp_distances ∗distances, int i, int j, float d)
- struct btp_distances ∗ btp_gdb_threads_compare (struct btp_gdb_thread ∗∗threads, int m, int n, btp_-dist_thread_type dist_func)

### 5.14.1 Detailed Description

Distance between stack trace threads.

### 5.14.2 Typedef Documentation

#### 5.14.2.1 typedef float(* btp_dist_thread_type)(struct btp_gdb_thread *, struct btp_gdb_thread *)

A function which compares two threads.

### 5.14.3 Function Documentation

#### 5.14.3.1 struct btp_distances* btp_distances_dup (struct btp_distances * *distances*) [read]

Creates a duplicate of the distances structure.

**Parameters:**

***distances*** It must be non-NULL pointer. The structure is not modified by calling this function.

**Returns:**

This function never returns NULL.

#### 5.14.3.2 void btp_distances_free (struct btp_distances * *distances*)

Releases the memory held by the distances structure.

**Parameters:**

***distances*** If the distances is NULL, no operation is performed.

#### 5.14.3.3 float btp_distances_get_distance (struct btp_distances * *distances*, int *i*, int *j*)

Gets the entry (i, j) from the distance matrix.

**Parameters:**

***distances*** It must be non-NULL pointer.

***i*** Row in the matrix.

***j*** Column in the matrix.

**Returns:**

For entries (i, i) zero distance is returned and values returned for entries (i, j) and (j, i) are the same.

#### 5.14.3.4 struct btp_distances* btp_distances_new (int *m*, int *n*) [read]

Creates and initializes a new distances structure.

**Parameters:**

***m*** Number of rows.

***n*** Number of columns.

**Returns:**

It never returns NULL. The returned pointer must be released by calling the function btp_distances_-free().

#### 5.14.3.5 void btp_distances_set_distance (struct btp_distances * *distances*, int *i*, int *j*, float *d*)

Sets the entry (i, j) from the distance matrix.

**Parameters:**

***distances*** It must be non-NULL pointer.

***i*** Row in the matrix.

***j*** Column in the matrix.

***d*** Distance.

#### 5.14.3.6 struct btp_distances* btp_gdb_threads_compare (struct btp_gdb_thread ** *threads*, int *m*, int *n*, btp_dist_thread_type *dist_func*) [read]

Creates a distances structure by comparing threads.

**Parameters:**

***threads*** Array of threads. They are not modified by calling this function.

***m*** Compare first m threads from the array with other threads.

***n*** Number of threads in the passed array.

***dist_func*** Distance function which will be used to compare the threads. It's assumed to be symmetric and return zero distance for equal threads.

**Returns:**

This function never returns NULL.

## 5.15 normalize.h File Reference

Normalization of stack traces.

### Functions

- void **btp_normalize_thread** (struct btp_gdb_thread *thread)
- void **btp_normalize_backtrace** (struct btp_gdb_backtrace *backtrace)
- void **btp_normalize_dbus_thread** (struct btp_gdb_thread *thread)
- void **btp_normalize_gdk_thread** (struct btp_gdb_thread *thread)
- void **btp_normalize_gtk_thread** (struct btp_gdb_thread *thread)
- void **btp_normalize_glib_thread** (struct btp_gdb_thread *thread)
- struct btp_gdb_frame * btp_glibc_thread_find_exit_frame (struct btp_gdb_thread *thread)
- void **btp_normalize_glibc_thread** (struct btp_gdb_thread *thread)
- void **btp_normalize_libstdcpp_thread** (struct btp_gdb_thread *thread)
- void **btp_normalize_linux_thread** (struct btp_gdb_thread *thread)
- void **btp_normalize_xorg_thread** (struct btp_gdb_thread *thread)
- void btp_normalize_paired_unknown_function_names (struct btp_gdb_thread *thread1, struct btp_gdb_thread *thread2)
- void btp_normalize_optimize_thread (struct btp_gdb_thread *thread)

### 5.15.1 Detailed Description

Normalization of stack traces.

### 5.15.2 Function Documentation

#### 5.15.2.1 struct btp_gdb_frame* btp_glibc_thread_find_exit_frame (struct btp_gdb_thread * *thread*) [read]

Checks whether the thread it contains some function used to exit application. If a frame with the function is found, it is returned. If there are multiple frames with abort function, the lowest one is returned.

**Returns:**

Returns NULL if such a frame is not found.

#### 5.15.2.2 void btp_normalize_optimize_thread (struct btp_gdb_thread * *thread*)

Remove frames which are not interesting in comparison with other threads.

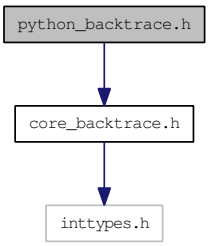
#### 5.15.2.3 void btp_normalize_paired_unknown_function_names (struct btp_gdb_thread * *thread1*, struct btp_gdb_thread * *thread2*)

Renames unknown function names ("??") that are between the same function names to be treated as similar in later comparison. Leaves unpair unknown functions unchanged

## 5.16 python_backtrace.h File Reference

Python stack trace structure and related algorithms. `#include "core_backtrace.h"`

Include dependency graph for python_backtrace.h:




### Functions

- struct btp_core_backtrace * **btp_core_python_parse_backtrace** (const char *text)

### 5.16.1 Detailed Description

Python stack trace structure and related algorithms.

## 5.17 sha1.h File Reference

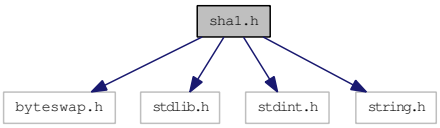
An implementation of SHA-1 cryptographic hash function. `#include <byteswap.h>`

`#include <stdlib.h>`

`#include <stdint.h>`

`#include <string.h>`

Include dependency graph for sha1.h:




### Data Structures

- struct btp_sha1_state

### Defines

- #define **BTP_SHA1_RESULT_BIN_LEN** (5 ∗ 4)
- #define **BTP_SHA1_RESULT_LEN** (5 ∗ 4 ∗ 2 + 1)

### Functions

- void **btp_sha1_begin** (struct btp_sha1_state ∗state)
- void **btp_sha1_hash** (struct btp_sha1_state ∗state, const void ∗buffer, size_t len)
- void **btp_sha1_end** (struct btp_sha1_state ∗state, void ∗resbuf)
- char ∗ **btp_bin2hex** (char ∗dst, const char ∗str, int count)

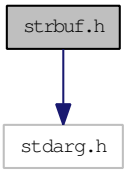
### 5.17.1 Detailed Description

An implementation of SHA-1 cryptographic hash function.

## 5.18 strbuf.h File Reference

String buffer structure and related algorithms. `#include <stdarg.h>`

Include dependency graph for strbuf.h:




### Data Structures

- struct btp_strbuf

### Functions

- struct btp_strbuf ∗ btp_strbuf_new ()
- void btp_strbuf_init (struct btp_strbuf ∗strbuf)
- void btp_strbuf_free (struct btp_strbuf ∗strbuf)
- char ∗ btp_strbuf_free_nobuf (struct btp_strbuf ∗strbuf)
- void btp_strbuf_clear (struct btp_strbuf ∗strbuf)
- void btp_strbuf_grow (struct btp_strbuf ∗strbuf, int num)
- struct btp_strbuf ∗ btp_strbuf_append_char (struct btp_strbuf ∗strbuf, char c)
- struct btp_strbuf ∗ btp_strbuf_append_str (struct btp_strbuf ∗strbuf, const char ∗str)
- struct btp_strbuf ∗ btp_strbuf_prepend_str (struct btp_strbuf ∗strbuf, const char ∗str)
- struct btp_strbuf ∗ btp_strbuf_append_strf (struct btp_strbuf ∗strbuf, const char ∗format,...)
- struct btp_strbuf ∗ btp_strbuf_append_strfv (struct btp_strbuf ∗strbuf, const char ∗format, va_list p)
- struct btp_strbuf ∗ btp_strbuf_prepend_strf (struct btp_strbuf ∗strbuf, const char ∗format,...)
- struct btp_strbuf ∗ btp_strbuf_prepend_strfv (struct btp_strbuf ∗strbuf, const char ∗format, va_list p)

### 5.18.1 Detailed Description

String buffer structure and related algorithms.

### 5.18.2 Function Documentation

#### 5.18.2.1 struct btp_strbuf∗ btp_strbuf_append_char (struct btp_strbuf ∗ *strbuf*, char *c*) [read]

The current content of the string buffer is extended by adding a character c at its end.

#### 5.18.2.2 struct btp_strbuf∗ btp_strbuf_append_str (struct btp_strbuf ∗ *strbuf*, const char ∗ *str*) [read]

The current content of the string buffer is extended by adding a string str at its end.

#### 5.18.2.3 struct btp_strbuf* btp_strbuf_append_strf (struct btp_strbuf * *strbuf*, const char * *format*, ...) [read]

The current content of the string buffer is extended by adding a sequence of data formatted as the format argument specifies.

#### 5.18.2.4 struct btp_strbuf* btp_strbuf_append_strfv (struct btp_strbuf * *strbuf*, const char * *format*, va_list *p*) [read]

Same as btp_strbuf_append_strf except that va_list is used instead of variable number of arguments.

#### 5.18.2.5 void btp_strbuf_clear (struct btp_strbuf * *strbuf*)

The string content is set to an empty string, erasing any previous content and leaving its length at 0 characters.

#### 5.18.2.6 void btp_strbuf_free (struct btp_strbuf * *strbuf*)

Releases the memory held by the string buffer.

**Parameters:**

***strbuf*** If the strbuf is NULL, no operation is performed.

#### 5.18.2.7 char* btp_strbuf_free_nobuf (struct btp_strbuf * *strbuf*)

Releases the strbuf, but not the internal buffer. The internal string buffer is returned. Caller is responsible to release the returned memory using free().

#### 5.18.2.8 void btp_strbuf_grow (struct btp_strbuf * *strbuf*, int *num*)

Ensures that the buffer can be extended by num characters without dealing with malloc/realloc.

#### 5.18.2.9 void btp_strbuf_init (struct btp_strbuf * *strbuf*)

Initializes all members of the strbuf structure to their default values. No memory is released, members are simply overritten. This is useful for initializing a strbuf structure placed on the stack.

#### 5.18.2.10 struct btp_strbuf* btp_strbuf_new () [read]

Creates and initializes a new string buffer.

**Returns:**

It never returns NULL. The returned pointer must be released by calling the function btp_strbuf_free().

#### 5.18.2.11 struct btp_strbuf* btp_strbuf_prepend_str (struct btp_strbuf * *strbuf*, const char * *str*) [read]

The current content of the string buffer is extended by inserting a string str at its beginning.

#### 5.18.2.12 struct btp_strbuf* btp_strbuf_prepend_strf (struct btp_strbuf * *strbuf*, const char * *format*, ...) [read]

The current content of the string buffer is extended by inserting a sequence of data formatted as the format argument specifies at the buffer beginning.

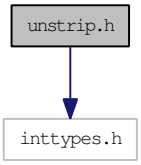
#### 5.18.2.13 struct btp_strbuf* btp_strbuf_prepend_strfv (struct btp_strbuf * *strbuf*, const char * *format*, va_list *p*) [read]

Same as btp_strbuf_prepend_strf except that va_list is used instead of variable number of arguments.

## 5.19 unstrip.h File Reference

Parser for the output of the unstrip utility. `#include <inttypes.h>`

Include dependency graph for unstrip.h:




### Data Structures

- struct btp_unstrip_entry

### Functions

- struct btp_unstrip_entry * **btp_unstrip_parse** (const char *unstrip_output)
- struct btp_unstrip_entry * **btp_unstrip_find_address** (struct btp_unstrip_entry *entries, uint64_t address)
- void **btp_unstrip_free** (struct btp_unstrip_entry *entries)

### 5.19.1 Detailed Description

Parser for the output of the unstrip utility.

## 5.20 utils.h File Reference

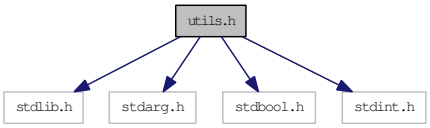
Various utility functions, macros and variables that do not fit elsewhere. `#include <stdlib.h>`

`#include <stdarg.h>`

`#include <stdbool.h>`

`#include <stdint.h>`

Include dependency graph for utils.h:




### Defines

- #define **BTP_lower** "abcdefghijklmnopqrstuvwxyz"
- #define **BTP_upper** "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
- #define **BTP_alpha** BTP_lower BTP_upper
- #define **BTP_space** " \t\r\n\v\f"
- #define **BTP_digit** "0123456789"
- #define **BTP_alnum** BTP_alpha BTP_digit

### Functions

- void ∗ btp_malloc (size_t size)
- void ∗ btp_mallocz (size_t size)
- void ∗ btp_realloc (void ∗ptr, size_t size)
- char ∗ btp_vasprintf (const char ∗format, va_list p)
- char ∗ btp_asprintf (const char ∗format,...)
- char ∗ btp_strdup (const char ∗s)
- char ∗ btp_strndup (const char ∗s, size_t n)
- int btp_strcmp0 (const char ∗s1, const char ∗s2)
- char ∗ btp_strchr_location (const char ∗s, int c, int ∗line, int ∗column)
- char ∗ btp_strstr_location (const char ∗haystack, const char ∗needle, int ∗line, int ∗column)
- size_t btp_strspn_location (const char ∗s, const char ∗accept, int ∗line, int ∗column)
- char ∗ btp_file_to_string (const char ∗filename)
- bool btp_skip_char (const char ∗∗input, char c)
- bool btp_skip_char_limited (const char ∗∗input, const char ∗allowed)
- bool btp_parse_char_limited (const char ∗∗input, const char ∗allowed, char ∗result)
- int btp_skip_char_sequence (const char ∗∗input, char c)
- int btp_skip_char_span (const char ∗∗input, const char ∗chars)
- int btp_skip_char_span_location (const char ∗∗input, const char ∗chars, int ∗line, int ∗column)
- int btp_parse_char_span (const char ∗∗input, const char ∗accept, char ∗∗result)
- bool btp_parse_char_cspan (const char ∗∗input, const char ∗reject, char ∗∗result)
- int btp_skip_string (const char ∗∗input, const char ∗string)
- bool btp_parse_string (const char ∗∗input, const char ∗string, char ∗∗result)
- char btp_parse_digit (const char ∗∗input)

- int btp_skip_unsigned_integer (const char **input)
- int btp_parse_unsigned_integer (const char **input, unsigned *result)
- int btp_skip_hexadecimal_number (const char **input)
- int btp_parse_hexadecimal_number (const char **input, uint64_t *result)
- char * **btp_skip_whitespace** (const char *s)
- char * **btp_skip_non_whitespace** (const char *s)

## Variables

- bool btp_debug_parser

### 5.20.1 Detailed Description

Various utility functions, macros and variables that do not fit elsewhere.

### 5.20.2 Function Documentation

#### 5.20.2.1 char* btp_asprintf (const char * *format*, ...)

Never returns NULL.

#### 5.20.2.2 char* btp_file_to_string (const char * *filename*)

Loads file contents to a string.

**Returns:**

File contents. If file opening/reading fails, NULL is returned.

#### 5.20.2.3 void* btp_malloc (size_t *size*)

Never returns NULL.

#### 5.20.2.4 void* btp_mallocz (size_t *size*)

Never returns NULL.

#### 5.20.2.5 bool btp_parse_char_cspan (const char ** *input*, const char * *reject*, char ** *result*)

If the input contains characters which are not in string reject, create a string from this sequence and store it to the result, move the input pointer after the sequence, and return true. Otherwise do not modify the input and return false.

If this function returns true, the caller is responsible to free the result.

#### 5.20.2.6 bool btp_parse_char_limited (const char ** *input*, const char * *allowed*, char * *result*)

If the input contains one of allowed characters, store the character to the result, move the input pointer after that character, and return true. Otherwise do not modify the input and return false.

#### 5.20.2.7 int btp_parse_char_span (const char ** *input*, const char * *accept*, char ** *result*)

If the input contains one or more characters from string accept, create a string from this sequence and store it to the result, move the input pointer after the sequence, and return the lenght of the sequence. Otherwise do not modify the input and return 0.

If this function returns nonzero value, the caller is responsible to free the result.

#### 5.20.2.8 char btp_parse_digit (const char ** *input*)

If the input contains digit 0-9, return it as a character and move the input pointer after it. Otherwise return '' and do not modify the input.

#### 5.20.2.9 int btp_parse_hexadecimal_number (const char ** *input*, uint64_t * *result*)

If the input contains 0x[0-9a-f]+, parse the number, and move the input pointer after it. Otherwise do not modify the input.

**Returns:**

The number of characters read from input. 0 if the input does not contain a hexadecimal number.

#### 5.20.2.10 bool btp_parse_string (const char ** *input*, const char * *string*, char ** *result*)

If the input contains the string, copy the string to result, move the input pointer after the string, and return true. Otherwise do not modify the input and return false.

If this function returns true, the caller is responsible to free the result.

#### 5.20.2.11 int btp_parse_unsigned_integer (const char ** *input*, unsigned * *result*)

If the input contains [0-9]+, parse it, move the input pointer after the number.

**Returns:**

Number of parsed characters. 0 if input does not contain a number.

#### 5.20.2.12 void* btp_realloc (void * *ptr*, size_t *size*)

Never returns NULL.

#### 5.20.2.13 bool btp_skip_char (const char ** *input*, char *c*)

If the input contains character c in the current positon, move the input pointer after the character, and return true. Otherwise do not modify the input and return false.

#### 5.20.2.14 bool btp_skip_char_limited (const char ** *input*, const char * *allowed*)

If the input contains one of allowed characters, move the input pointer after that character, and return true. Otherwise do not modify the input and return false.

#### 5.20.2.15 int btp_skip_char_sequence (const char ** *input*, char *c*)

If the input contains the character c one or more times, update it so that the characters are skipped. Returns the number of characters skipped, thus zero if **input does not contain c.

#### 5.20.2.16 int btp_skip_char_span (const char ** *input*, const char * *chars*)

If the input contains one or more characters from string chars, move the input pointer after the sequence. Otherwise do not modify the input.

**Returns:**

The number of characters skipped.

#### 5.20.2.17 int btp_skip_char_span_location (const char ** *input*, const char * *chars*, int * *line*, int * *column*)

If the input contains one or more characters from string chars, move the input pointer after the sequence. Otherwise do not modify the input.

**Parameters:**

***line*** Starts from 1. Corresponds to the returned number.

***column*** Starts from 0. Corresponds to the returned number.

**Returns:**

The number of characters skipped.

#### 5.20.2.18 int btp_skip_hexadecimal_number (const char ** *input*)

If the input contains 0x[0-9a-f]+, move the input pointer after that.

**Returns:**

The number of characters processed from input. 0 if the input does not contain a hexadecimal number.

#### 5.20.2.19 int btp_skip_string (const char ** *input*, const char * *string*)

If the input contains the string, move the input pointer after the sequence. Otherwise do not modify the input.

**Returns:**

Number of characters skipped. 0 if the input does not contain the string.

#### 5.20.2.20 int btp_skip_unsigned_integer (const char ** *input*)

If the input contains [0-9]+, move the input pointer after the number.

**Returns:**

The number of skipped characters. 0 if input does not start with a digit.

#### 5.20.2.21 char* btp_strchr_location (const char * *s*, int *c*, int * *line*, int * *column*)

A strchr() variant providing line and column in the string s indicating where the char c was found.

**Parameters:**

***line*** Starts from 1. Its value is valid only when this function does not return NULL.
***column*** Starts from 0. Its value is valid only when this function does not return NULL.

#### 5.20.2.22 int btp_strcmp0 (const char * *s1*, const char * *s2*)

A strcmp() variant that works also with NULL parameters. NULL is considered to be less than a string.

#### 5.20.2.23 char* btp_strdup (const char * *s*)

Never returns NULL.

#### 5.20.2.24 char* btp_strndup (const char * *s*, size_t *n*)

Never returns NULL.

#### 5.20.2.25 size_t btp_strspn_location (const char * *s*, const char * *accept*, int * *line*, int * *column*)

A strspn() variant providing line and column of the string s which corresponds to the returned length.

**Parameters:**

***line*** Starts from 1.
***column*** Starts from 0.

#### 5.20.2.26 char* btp_strstr_location (const char * *haystack*, const char * *needle*, int * *line*, int * *column*)

A strstr() variant providing line and column of the haystick indicating where the needle was found.

**Parameters:**

***line*** Starts from 1. Its value is valid only when this function does not return NULL.
***column*** Starts from 0. Its value is valid only when this function does not return NULL.

#### 5.20.2.27 char* btp_vasprintf (const char * *format*, va_list *p*)

Never returns NULL.

### 5.20.3 Variable Documentation

#### 5.20.3.1 bool btp_debug_parser

Debugging output to stdout while parsing. Default value is false.

# Chapter 6

# Known Bugs

Empty.

# Index